SLAM-CLAM Specification

31319

Revision History

Rev	Date	Purpose	Originator
Draft	2003-04-17	First Draft	dmagley
0.2	2003-04-25	Added Objectives and router/subscriber CM flows	dmagley
0.3	2003-06-11	Added High level design section and PLM modules	dmagley
0.4	2003-7-17	Added Document number and SLAM Design section	dmagley, tgupta
0.5	2003-8-1	Added sections on Call capture, sigdbg, Misc. SLAM and CLAM stuff, IP forwarder, CCSid Assignment, TL1 FSM and base CLAM design section.	dmagley
0.6	2003-9-17	Clean up diagrams and flows for CCM, CCS Assignment, DB Manager now on CCM and not a SLAM/CLAM, SLAM to CLAM API also added.	dmagley
0.7	2003-10-10	Changes for ServiceChange	M.Pralat
0.8	2005-2-18	General cleanup after reviewed.	dmagley
0.9	2006-11-17	Added to QDI…new number	dmagley
0.10	2007-02-12	Added section on LM rewind and 6-3 design for staged rewind	ramya

Table of Contents

1 Introduction

1.1 Scope

This document describes the functional and design specifications of System Layer and Alarm Manager (SLAM) and Call Control Layer and Alarm Manager (CLAM) software in the Telica Plexus 9000. This will be utilized in release 5.0 and later. The SLAM will communicate with multiple CLAMs, and SLAMs and interface with the database manager, Equipment manager and TL1 agent as specified in [1]

1.2 Target Audience

- Component Developers for this component.
- Software Unit Testers
- Component developers for components using this component services

1.3 References

1) 79-5017, Media Gateway Controller (MGC) Architecture Specification
2) 79-3139, TL1 Commands Functional Specification for Subscriber and Routing Database
3) 79-3311, Compute Module Monitor Software Specification
4) 79-3326, PLM/CLAM ServiceChange Handling, M. Pralat
5) 79-3322, MGC IP Forwarding Functional Specification, D. Dafoe

2 System Requirements:

1. TL1 needs to be able to handle multiple outstanding transactions since many operators will be on the switch at same time. Therefore the SLAM/CLAM must be able to handle multiple requests at the same time.
2. MGC should be able to handle 50 transactions per second.
3. The working SP will provide all FAM support. Each CM/CCS/CLAM will send alarms to the SP with enough information to generate an intelligent alarm/event to help the user understand what went wrong.
4. All router and subscriber provisioning validation will be done on the CM they are running on. Therefore the SLAM will require the Router and subscriber CM to be operational before the provisioning of these parameters.
5. The SLAM must be designed to handle 512 CLAMs (8 CLAM/CPU* 4 CPU/CM *16 CM).

   Issue 1: At what point does a CM failover to protection? If there are 4 CCS on a CM, and 3 out of 4 are operational with a HOT standby. The 4th is still syncing. Do we fail over the whole CM if any of the CCS has a fatal error? Answer: NO. The only time we fail over, is when all are synced up. We can not fail over since the non Synced CCS will not be able to fail over and will be down. All calls on the CCS may not work. So only if all are synced do we fail over. Otherwise we just restart the failed CCS.

6. In the distributed version, the router and subscriber applications will run on a special CM that contains a disk and times ten application. This is needed to hold these large databases. In the integrated version it will all run on the SP.
~~7.Must be able to perform a hot upgrade of the CM or SP from a previous version. Also need upgrade from 3.9 to this new model.~~
~~Will this be required to be a HOT upgrade? Answer. Yes must be hot upgradeable. The tough part will be to sync and replicate from the old PLM to the new CLAM.~~

~~8.~~7.A fail over of the SP during a TL1 provision, must not cause system corruption. The command can time out is allowed. But if the command completes successful, then the TL1 shall receive the correct response. Alarms must not be lost.

~~9.~~8.The new Layer management design must interact with an MG via Megaco service changes.

~~10.~~9. We want to be able to configure the MGC to operate without a PointCode (or use a dummy PointCode is not exposed to the outside world if a real PC is not configured). This is to support future configurations that are just doing IP or CAS or ISDN with no SS7. Specifically, the MG may use the on-board MGC to do GR303+CAS+MGCP to E911(CAS) calls.

~~11.~~10. We need to support many up to 512 DPC. Not every CCS process has to support all DPCs. Need to design system so that a DPC is only configured on a CCS process when that CCS process is configured to talk to that DPC (either ISUP or TCAP).

~~12.~~11. Before an MG-ASSOC can be deleted, the SLAM must validate that all items that are provisioned for that MG will be deleted first.

3 Design Objectives:

1. The SLAM will validate all requests. ~~other than routing and subscriber.~~ This is to reduce the response time. This includes the checks in the protocol layers.
2. The integrated application will run 2 PLMs on the SP. The original PLM can handle both the SG and MG applications and the new SLAM will handle the MGC.
3. RTRV's that are required to go to the CLAM will be done with multi-gets to minimize the latency.
4. Rewind commands (where the SLAM is provisioning the CLAM after a bootup) require some type of bulk or multi messages to speed up the process.
5. Replication of database will be done by the protection SLAM. The original idea was to have TimesTen do the replication. The Active SLAM will need to send state info to the protection SLAM in order to handle SP failovers with outstanding CLAM requests. Since this is going to the protection, might as well use the existing replication code where the PLM replicates to the standby.

6. The SLAM will need to replicate per message state info to the protection SLAM. This is to ensure consistency if the SP fails over to see if any outstanding commands are waiting. When the Active SLAM sends a REQ to the CLAM, it will also send a message with the CLAM transaction ID to the standby SLAM. When the Active SLAM gets a response from the CLAM, it will send a transaction ACK to the standby SLAM to clear up the FSM. If the SLAM fails

over to the Standby, the CLAM will then send the ACT directly to the Standby (New active) and the SLAM will clean up the FSM as required.
~~Issue 3:How should this audit be handled?~~

~~a: could just replay everything the CM knows to the SLAM (problem is this could be huge)~~
~~b: could use sequence numbers in SLAM/CLAM messages and audit only the missing sequence numbers.~~
~~c: could use an update flag in the database that gets set when ACK is sent from CLAM. How do we handle the alarms and events?~~
~~d: Have the active SLAM replicate to the standby slam the state info for each command sent down.~~ ~~This is the Present method that we will use.~~

7. Provisioning changes will be validated by SLAM, written to disk, and then the result returned to the TL1 agent. Only then will the info be sent to the CLAM. If the CLAM or lower applications fail the request, an alarm will be sent out.
8. Need to workout the upgrade path. Should be able to handle the SLAM first then all the CLAMs later. That will require an API that includes a version tag in the message header. The SLAM will have to send down messages formatted with what the CLAM expects to see.
9. Need to convert the existing TSMC and PMI support to PLM. This is about 10% and will not support a multi-session TL1 design.
10. When a CCS fails and needs to fail over to standby, the SLAM must tell all CCS of the change so relay procIds can be updated. The SLAM will have to keep state information on each CCS. If running in simplex mode and one CCS fails, then want to restart that CCS since cannot fail over.
11. New termination ID's are required. These will include the MG name. The PLM will need to parse this from the TL1 messages and generate the new data structures used in signaling. <MGNAME>-<IOM#>/<T1E1#>/<Chan#>
12. CLAM will have to receive service state changes from the MG via the Megaco and convert them into port state changes. Also IOM relay states and board states will need to be converted to a Megaco state.
13. For the SG, the original PLM concept will be used.
14. For the MG, the original PLM concept will be used.
15. The CLAM will need to configure RELAY with a process ID, slot and CPU ID. CMMON will start a CCS with the process ID.
    ~~**Issue 4:**~~**Issue 2:** This will be done independently of the CLAM. But it needs to be done with the infrastructure section of the CM. Clam may need to keep a list of the process IDs, slot and CPU ID for communication based on MGName.
16. The CLAM (or something else) needs to handle the infrastructure setup for the CCS process. This includes makefiles, starting of threads, relay startup…
17. SLAM needs to support a distribution function with the MGName. This name will be added with strings in ENT-MG and termination Ids. This needs to be converted to a numeric based scheme for use by the rest of the system.
18. There should only be one outstanding command sent to a CCS. This can be implemented in with a queue for each CCS.

    Note: This queue will naturally reside in the CLAM's relay queue. The CLAM will read out the cmd and forward it into the matrix and modules. There is already a natural

queue there where a module will block until complete so the cmd will be queued up.

19. When a call fails from the MG because of a mis-provisioning between the MGC and MG, an alarm should be set. The CLAM should update the RM and layers to say that T1 is not available. This alarm can be cleared when the ds0 goes back IS. This can be cleared up when the MG has been provisioned and then a TL1 command (INIT-DS0::…) is used to re-initialize this on the MGC.
20. The CLAM will handle the relay support to other processes in the CCS suite. There is one process per CPU called the Xmit Unit (XU) that will interface the 8 CCS's to the rest of the world.

4 Work Items:

This will be broken out to major section to help assign people to the tasks.

1. Modifications to existing PLM for the SG and MGC
   - New SG SYS support for SG
   - New MG SYS support for MG.
   - New MTP3/MTP2/M3UA support for SG.
   - Also need to support the SMDI in the original PLM
   - Convert TSMC to PLM. This would need to be done for both PLM and new SLAM/CLAM. Less so for the S/C since most of it is for MG? This includes removing all PMI support for the MTP3 and MTP2 configuration.

   ~~Issue 5:This bullet should be done last. It should not hold up a demo of an SG.~~

2. Split up PLM into SLAM and CLAM (High Level Design)
   - Core SLAM functions data base packing (plmRmtDataLoad() and plmRmtDataLoadClamId())
   - Additions to Matrix structures for validation function pointer and flags.
   - Core functions for CLAM (plmRmtDataRead())
   - Make sure the old PLM and new SLAM/CLAM can operate at same time for an integrated chassis. This should be with and without a CM or CCM (centralized CM).
   - Setup make files for process break up. The following is a list of processes.
     - For SG:
       - Need a signaling with M3UA, MTP3. (note will be run with the MG-sig to access the PLM support)
       - Separate MTP2
     - For MG:
       - Signaling with old PLM, CCRM, SFI …
     - For MGC:
       - Separate SLAM
       - CCS (signaling with CLAM, GCC,…)
       - Router (with own CLAM)
       - Subscriber (with own CLAM)

- Define API between SLAM and CLAM including bulk retrieves, bulk sets, and structure version numbers
- CVS and build support. Need new directory structure and make files

3. SLAM Control
   - State machine to support multiple messages sent to multiple CLAMs. This is needed to support the flows discussed later.
   - Need a new CM board state table
   - Need a new CCS state table. This will be used with EQM interface.
   - Need to track the Relay state for each CCS. If XU is up, SLAM marks all 8 CCS relay up.
   - Boot up support for CCS.

4. SLAM Configuration
   - Module based validation support
   - Module based Database access and packing of data to send to the CLAM
   - Database replication. Will it be the same message sent to the protection SLAM or CLAM, or will it be a PSF command?
   - Removal of any remaining PMI support.
   - Support converting MGName for the system. This includes doling out M3UA ports when a MG is added, and converting Termination Ids to a numeric structure.
   - Need to support the 3.9 Bulk download feature. This could fit in with the SLAM, where the validation is done for each item in the list, then the entire contents stored in the DB. Then we can send the data to the CLAM. This should fit nicely into the SLAM/CLAM design.

5. Alarms and Events
   - Alarm control. Need to design alarm handler and 2 level throttling for both SLAM and CLAM

6. CLAM Control
   - New module to interface with the MGI/MEGACO for configuration and service change events.
   - Convert Megaco Service changes populate the port state table
   - Relay support (process IDs)
   - Billing and stats support for each CCS. The only issue here may be “how do we get the billing server IP address to each GDI on the CCS.
7. CLAM Configuration
   - Module based changes to existing code to support each provisioned command.
   - New modules are required:
     - Module for BICC proxy?
     - Module for SIP proxy?

8. Upgrade support

- Need to support an HOT upgrade from 3.9 to an integrated chassis running the new architecture.
- Need to design slam/clam API to allow hot upgrades in the future. This is more apparent that before since the SLAM and CLAM are so closely designed with common matrix and structures.
- Work is needed to migrate from an integrated USA chassis to an MGC. The tough part may be to move the Router and Subscriber processes to the CCM. The CCS on the SP will be deleted as the new MG is added that contains the support for the previous integrated MG.

5 PLM Modules:

This table will indicate the existing PLM modules and on what platform it is needed. Items in the SG and MG column will run the older style PLM. Items in the SLAM, CLAM, Router/Subscriber and Service change are using the new split design.

Index	5.1.1.1 ENUM	5.1.1.2 Description	SG	SLAM	CLAM	MG	Router/ Subscriber	Service Change
0	PLM_CORE		X	X	X	X	X	
1	PLM_REP	Replication of some areas in TSMC	X		X	X	X	
2	PLM_TGP	SS7 TGP and CAS trunkgroup module		X	X		X	
3	PLM_IMT	SS7 Trunks		X	X			X
4	PLM_CAS	CAS DS0 module		X	X	X		X
5	PLM_SIPGN	SIP General Config module		X	X			
6	PLM_SIPADR	SIP-IPADDR module		X	X			
7	PLM_M3AC	M3UA-Client module		X	X			
8	PLM_M3AS	M3UA-Server module	X					
9	PLM_CPF	CAS profile module		X	X	X		
10	PLM_CIF	CAS interface module		X	X		X	
11	PLM_CGN	CAS general config module			X	X		
12	PLM_IOF	IOM failover module	X			X		
13	PLM_OLI	Originating Line Information screen module					X	
14	PLM_SWTCFG	Switch Config module (FIC/SCPI)		X	X			
15	PLM_BILLINGSYS	Billing System module (should be combined with BILLHOST)		X	X			
16	PLM_IVRSTATE	Ivr State						
17	PLM_GR3GEN	GR303 gen cfg module		X	X			
18	PLM_GR3IF	GR303 Interface module		X	X			
19	PLM_GR3CHAN	GR303 Chan module		X	X			X
20	PLM_GR3CRV	GR303 crv module		X	X		X	
21	PLM_FCFG	FIC General Config module		X	X			
22	PLM_FSGEN_CFG	Tufs Gen Cfg		X	X			
23	PLM_HNPA	Home NPA		X	X		X	
24	PLM_CARRIER	CARRIER		X	X		X	
25	PLM_SUBIF	SUBSCRIBER INTERFACE		X	X		X	
26	PLM_SCRNLST	screenList		X	X		X	
27	PLM_STST	STAT STAT						
28	PLM_AIN	Ain		X	X			
29	PLM_SERVICE	SERVICE		X	X			
30	PLM_LNPSCRDIG	LNP Screen Digits		X	X			
31	PLM_NNPA	National NPA		X	X		X	
32	PLM_HNXX	Home NXX		X	X		X	
33	PLM_OWNLRN	OWN LRN Config Module		X	X			
34	PLM_TOLLFREENPA	TOLLFREE NPA Config Module		X	X			
35	PLM_LNPADMIN	LNP ADMIN Table Config Module		X	X			
36	PLM_SUB	FIC Subscriber configuration		X	X		X	

Index	5.1.1.1 ENUM	5.1.1.2 Description	SG	SLAM	CLAM	MG	Router/ Subscriber	Service Change
37	PLM_LCC	FIC LCC configuration		X	X		X	
38	PLM_RC	FIC Rate centers		X	X			
39	PLM_CLNGAREA	FIC Calling areas		X	X			
40	PLM_COUNTRYCODE	Country Code		X	X		X	
41	PLM_SPDDGT	Speed Digit		X	X		X	
42	PLM_AINASS	Ain Assignment		X	X			
43	PLM_DR	DR(ALT_DN)		X	X		X	
44	PLM_SCRDGT	Screen Digits		X	X			
45	PLM_RELCAUSE	Release Cause		X	X			
46	PLM_BILLINGPRFL	Billing profile		X	X			
47	PLM_SUBSCOT	Subs Cot		X	X		X	
48	PLM_BILLSERV	Billing DGS Host Server Cfg		X	X			
49	PLM_SUBSCFV	SubsCFV		?	?		X	
50	PLM_MULTI_TL1	RtrvMulti	X	X	X		X	
51	PLM_IUAGEN	IUA general module		X	X	X		
52	PLM_IUAEP	IUA ASP EP module		X	X	X		
53	PLM_IUAASP	IUA ASP module		X	X	X		
54	PLM_IUAAS	IUA AS module		X	X	X		
55	PLM_IUALNK	IUA LNK module		X	X	X		
56	PLM_ING	Intelligent Network General Config		X	X			
57	PLM_CRS	Cross Connect or Nailed Up DS0				X		
58	PLM_AUTHCODELIST	Auth Code List module		X	X		X	
59	PLM_CCCLIST	CCCList module		X	X			
60	PLM_CDC	CDC module		X	X			
61	PLM_LAESCASE	LAES-CASE module		X	X			
62	PLM_CALEA_PM	Calea Pm module		X	X			
63	PLM_SIP_PM	SIP PM		X	X			
64	PLM_SS7SYS	SS7SYS module (SGSYS, MGCSYS)						
65	PLM_FTH	FTHA module	X	X	X	X	X	X
66	PLM_NDL	Nodal modle	X	X	X			
67	PLM_EVQ	Boot event queue module	X	X	X	X	X	X
68	PLM_NIF	NIF module	X					
69	PLM_MGN	MgcSys module		X	X		X	
70	PLM_SGN	SgSys module	X					
71	PLM_RTS	Nodal Router module		X	X		X	
72	PLM_MGCP_GEN	MGCP Gen module		X	X		X	
73	PLM_MGCP_SYS	MGCP system module		X	X			
74	PLM_MGCP_LNGP	MGCP Line Group module		X	X		X	
75	PLM_MGCP_GW	MGCP Gateway module		X	X			
76	PLM_MGCP_LINE	MGCP Line module		X	X			
77	PLM_MGCP_PM	MGCP PM module		X	X			
78	PLM_ACBAR	ACBAR		X	X			
79	PLM_SERVICE_DN	Remote Access to Call Forwarding		X	X		X	
80	PLM_IGN	ISDN general config module		X	X			

Index	5.1.1.1 ENUM	5.1.1.2 Description	SG	SLAM	CLAM	MG	Router/ Subscriber	Service Change
81	PLM_IIF	ISDN interface module		X	X		X	
82	PLM_ILK	ISDN link module		X	X	?		
83	PLM_IBC	ISDN B channel		X	X	X		X
84	PLM_LAPD	LAPD shared by Q931, GR303 and IUA				X		
85	PLM_SDP	Session description protocol profile table support		X	X			
86	PLM_VMSYS	VMS-SYS module		X				
87	PLM_VMSLNK	VMS-LNK module		X				
88	PLM_VMSCKT	VMS-CKTID module		X				
89	PLM_VMS_PM	VMS PM module		X				
90	PLM_SS7PC	SS7PC module (ISUP/BICC/TCAP…)		X	X			
91	PLM_DGT_MOD	digit modification module		X	X		X	
92	SLAM_CM_FAILOVER	to be defined		X	X			
	MG_SYS	new: MG sys for CCRM…				X		
	PLM_SS7PC	SS7PC module (MTP3, M3UA)	X					

6 Functional Design of SLAM/CLAM

6.1 Summary:

The existing PLM design needs to be modified for the new MGC architecture. There are different requirements for the SG, MG, SLAM, CLAM and the version running on the CM that will host the subscriber and router subsystems.

6.2 ~~Existing~~ Pre 5.0 Design:

The existing PLM comprises of 2 finite state machines (FSM) to handle the events. The top level will decode a given event and feed it into a 'matrix' of handlers. The matrix is a list of modules that need to handle the event. The state machine controls the flow through the various modules and if the event is a TL1 command, it sends the response back to the tl1 agent. Each module validates the command/event before feeding it to the second state machine. The first FSM and validation are stored in the plm files as plm_*module*1.c (i.e. plm_cas1.c).
Inside of each module of the matrix is the second state machine. This will handle the actual configuration and implementation of the command/event. The PLM supplies a common FSM design to be used by every module to ensure conformability. The 2nd FSM is contained in the plm_*module*2.c files (i.e. plm_cas2.c).

6.3 5.0 and Later ~~New~~ Design:

The PLM will need to be broken out in the MGC to support the SLAM and the CLAM division of labor since the database access will be on a separate board. The high level concept is to have the SLAM do all the validation and data base access and the CLAM to the actual implementation. This breaks out basically as the SLAM will contain the first

FSM and validation and the CLAM will handle the second. The event sent to the CLAM will be the same event that was sent to the SLAM. The question is how to do this. Here are the 2 options.

6.3.1 Option 1-Client/Server Model:

The term Client and Server do not really fit here, but the idea does. This model will allow the same code to co-exist for both the SLAM (client) and CLAM (server). At a high level, SLAM mode will be executed when the SLAM mode is set and the CLAM code will be run when operating in a CLAM mode. If both modes are set, then the PLM will operate as it does today (both will run). This will work for the SG and MG systems.
Advantages:

- Generic model that fits the SG, MG, SLAM/CLAM and the router/subscriber layer manager. The design for the SG and MG will be the same as for the MGC.
- Use same model that is implemented for other client/server processes like M3UA.
- Shorter time to market.

Disadvantages:

- Less readability. The code will not have a definite segregation of responsibilities. A developer will have to understand both the SLAM and CLAM sides of code.
- May be less efficient. The CLAM will run the same event decoder FSM as was run on the SLAM for each event. The SLAM could have a simpler setup.

6.3.2 Option 2-Separate Validation and Implementation Model:

This model will split apart the code base for the SLAM and CLAM. A new generic FSM will be created to process events on the SLAM that will only handle validation, database data collection and forwarding to the CLAM. This is much simpler than the one required on the CLAM since all validations will be moved from multi-modules to a single function. The existing event/matrix decode will be shared with the CLAM. New fields will be added to the matrixes that are for the SLAM only.

The CLAM will basically be the same as existing PLM. The differences are that the validation section is removed and the data required will be in the event sent from the SLAM (no second dipping of the TimesTen database). There will also be some data caching required to support service state changes.

Advantages:

- Allows code to be separated on a functional basis. Much easier to read and maintain.

Disadvantages:

- More time to implement. I estimate this is about a week more work per module and there are 100 modules. This will require many developers working in parallel.

Option 2 was implemented for all of 5.0 and later designs. .

6.3.3 SLAM Effort for Option 2:

Framework:

- Build the generic FSM to process events on the SLAM.
    - Validation
    - DB-access
    - Replicate the event to the standby SLAMs. This includes CLAM responses.
    - Send the event to CLAM
    - Handle TL1 response in the correct order (hold off response if another SLAM involved)
- Handle CLAM boot up
- New DB handling functions (generic packing, segmentation, sequence numbers etc)
- Manage CLAM state table. Hold off configuration while active OR standby CLAM booting up and syncing up.

Per event/matrix:

- Modify each matrix in plm_bdy1.c to add the SLAM specific attributes. These attributes define the validation/DB-access function(s) for that event and the order of processing this event.
- write the validation/db-access functions (dbproxy) for each of the event categories (matrices), approx 400 as it stands now - a lot of them would be re-usable. This functionality is mostly cut-and-paste from the existing PLM module file-1.
- write boot up functions, there is a boot up matrix already, boot up DB-accesses from all other modules will have to be cut-paste in here.

6.3.4 CLAM Effort for Option 2:

Framework:

- write DB handling (unpacking, reassembly, sequence numbers etc)
- Call existing FSM.
- Handle sending responses and alarms to SLAM

Per matrix/module:

- Remove validation sections in module file1, DB access calls modified to access data coming down from the SLAM
- additional work to manage in memory tables required to handle certain events autonomously on the CLAM
- Remove board and port events and add support for new service change messages received from the MG via MEGACO.

6.4 Basic TL1 Design Flow

1. The existing event decode and matrix concept will remain for both the SLAM and CLAM. This involves receiving a GIT message and sending it to plmProcessTL1Evt() which feeds into plmFindEvMt() which builds a key and

hashes into the event matrix table. The actual event is fed into the detected matrix with plmRaiseEvt().

2. The matrix will be modified to add in new SLAM parameters. This will basically be a function pointer to handle the validation and pack up the data base rows required for the CLAM. Also include flags to control the flow from the SLAM to other SLAMs or CLAMs as required.
3. So once a matrix is given an event (plmRaiseEvt()), if the SLAM flag is set, then the SLAM FSM is started. This functionality will be moved from the present plm*Module*bdy1.c file to a file with a prefix of 'sl' to handle the following events:
    a. validate command
    b. handle DB access
    c. pack DB data into a CLAM message
    d. return TL1 response if needed
    e. send message to CLAM or other SLAM if needed
    f. wait for CLAM response or time out
    g. send message state to protection SLAM (to handle case where the SLAM SP fails over once a message is sent to the CLAM, but before a response comes back) glorious
4. On the CLAM, the same decode will occur with step 1. The GIT message and using a key to find the correct matrix. Basically the existing plm*Module*bdy2.c files will be renamed to cl*Module*.c. This contains the provisioning FSM support.
5. Changes will be made to the new module use the new DB packed message sent from the SLAM instead of using the context info sent in with a FSM call in the old PLM design.
6. There are also multiple modules that actually read directly from the data base. These need to be identified and moved to the SLAM to be send down in the event message.
7. Also changes are needed to remove the board and port state changes and replace them with Megaco service changes.

7 CVS Directory and File Ideas

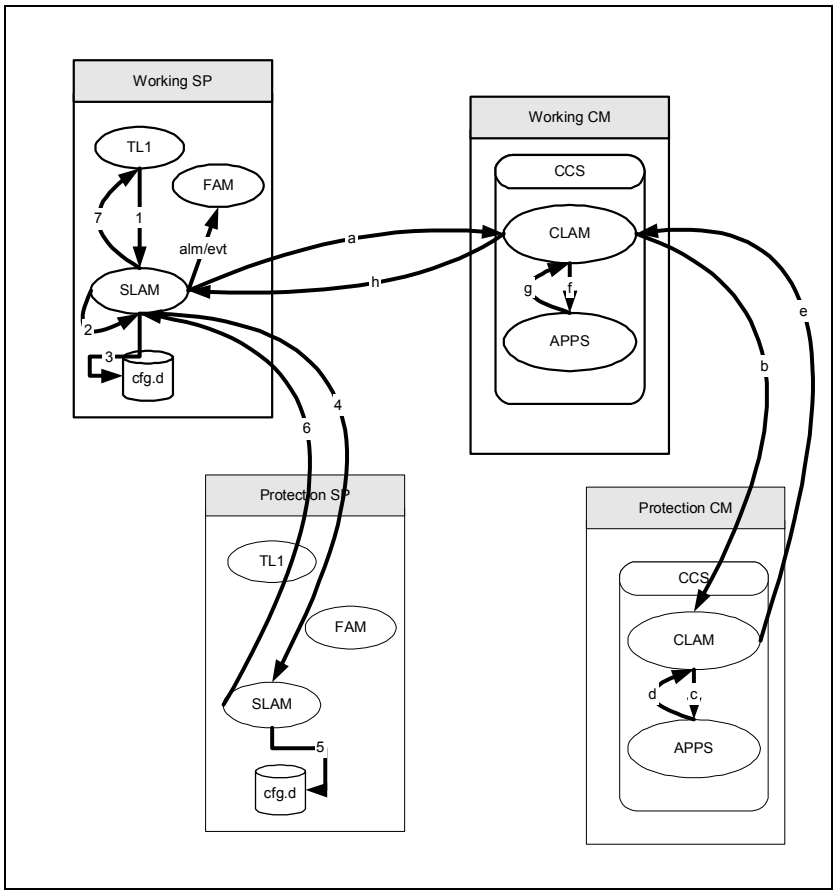
One goal is to have the SG and MG use the older PLM design that is shipping with 3.8 and 3.9 systems. These will reside in the existing signaling/tsm/tsmc directory with the existing file names. There will be a new directory structure to store the new design. This will be signaling/lm/. The sub directories will be as follows:

- signaling/lm/common
    - common matrix files
    - core functions
    - structure definitions
    - database access functions to fill up message structures to send to the clam
- signaling/lm/slam
    - system FSM that will handle flow of database calls and message passing to the clam
    - validation functions
    - file and function names should start with 'sl'
- signaling/lm/clam

- existing plm files modified not to do DB access or validation
- file and function names should start with ‘cl’

8 TL1 Command Flows

8.1 ENT and ED Commands



1. TL1 ENT or ED command comes to SLAM
2. SLAM Validates provisioning
3. Update working database.
4. Update protection SP's database either by TimesTen replication or a direct command from the working SLAM.
5. Update the protection database
6. Receive ACK/NACK from protection SP's database.
   ~~Issue 6:Does TimesTen supply a response indication when complete with the replication?~~

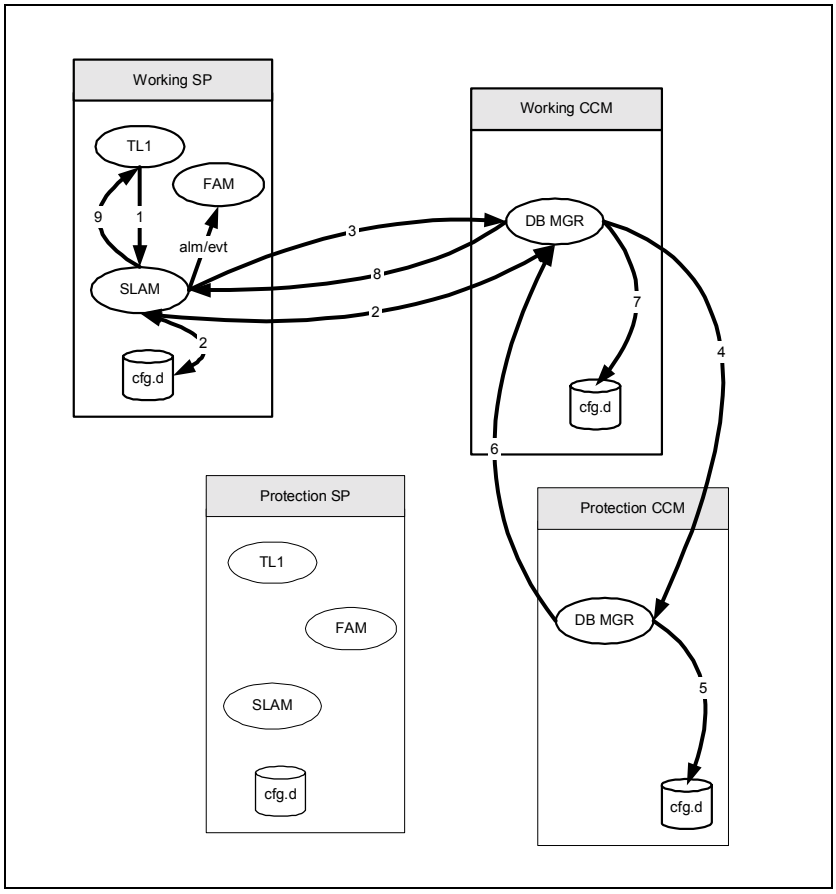
   ~~Issue 7:SLAM on protection needs the info. Either a callback from times ten or we do the old style of protection SLAM updates the database~~ ~~Go with old style~~
7. Return results to TL1. If protection update responded with a NACK, then raise an alarm and reboot protection.

a. 2nd state machine started after step 7. Send command to active CLAM. No validation done at the CLAM.
b. Send command to protection CLAM
c. Send command to applications.

d. Get ACK/NACK from applications.
e. Send ACK/NACK back to working CLAM.
f. If ACK, send to applications, If NACK, send alarm to FAM, and send to applications.
   ~~**Issue 8:**~~**Issue 3:** Do we reboot the protection CM if NACK? *NO….active will probably fail.*
g. CLAM gets an ACK or NACK from apps
h. CLAM sends working SLAM ACK/NACK. If NACK, send alarm to FAM.

- If any of the CM's or protection SP are OOS, then the states associated with them will be bypassed by SLAM state machine.

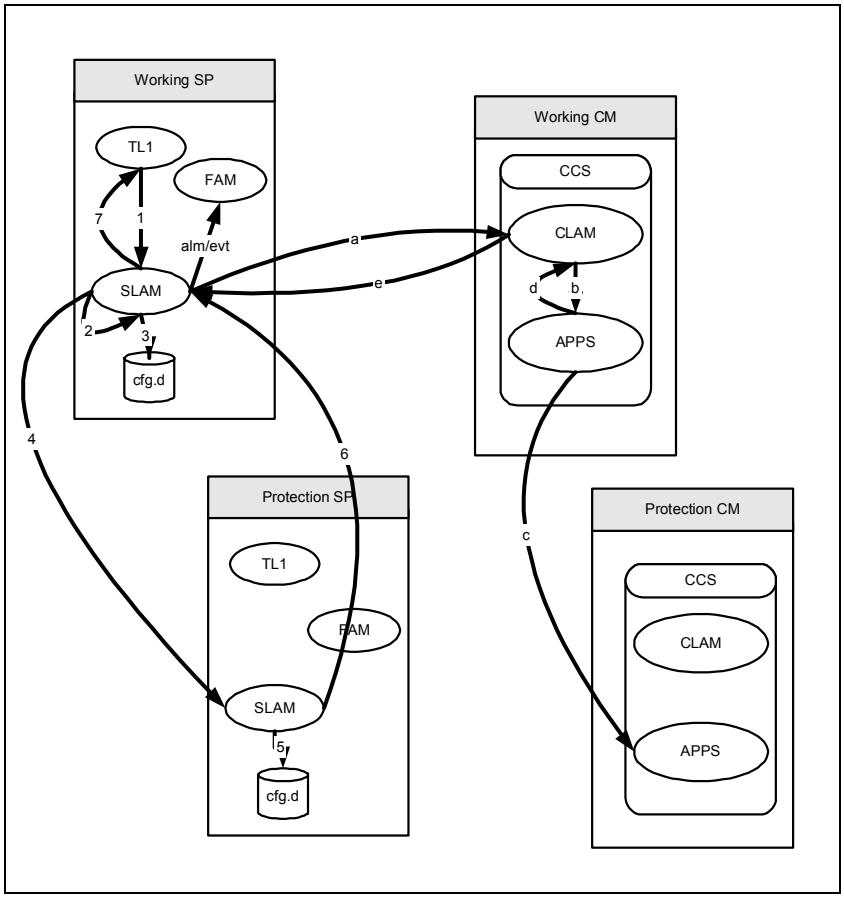
8.2 ENT and ED Commands (Subscriber DB)



This flow differs from a normal MGC ENT/ED command because the SP SLAM does not do the storage. This is done on the CCM module running the subscriber application.. The protection will also need to be configured. The db manager will use it's own replication design and will send the same command to the protection first then to the working. This is similar to the standard CLAM replication model. This is done to validate it on the standby first as an extra test. If it passes, it will most likely pass on the active.

1. TL1 ENT or ED command comes to SLAM
2. Validation is done on the SLAM. Note that may need to read from the local DB and even from the subscriber database on the CCM for validation.
3. SLAM does a DBWrite to store the data on the remote DB. This is sent to the working CCM.
4. The working DB Manager will first forward the write request to the protection DB Manager.
5. Protection DB Manager writes the data.
6. Send ACK/NACK to the working DB Manager
7. Working DB Manager writes the data.
8. Return status to SP SLAM.
9. Return status to TL1

8.3 DLT Commands with Trillium Control Request



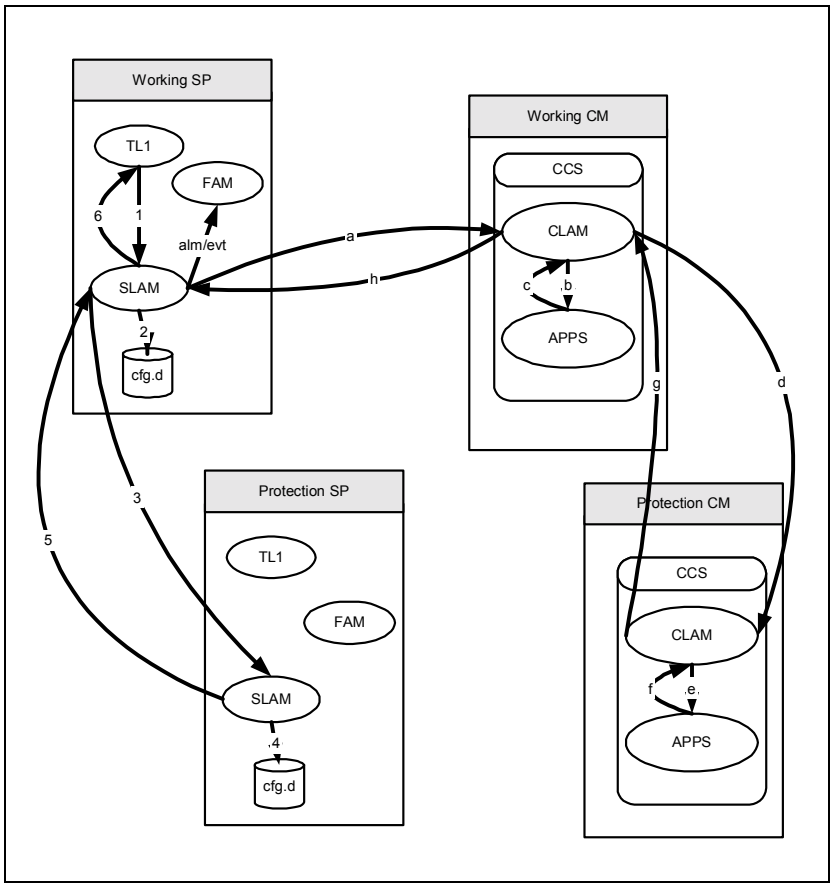
This section is for original Trillium applications like ISUP where the delete is handled in a control message. The control messages replicate with a PSF command.

1. TL1 DLT command comes to SLAM
2. SLAM Validates provisioning
3. If an ACK, commit change to database. If NACK, send error response to TL1.
4. Standby SP is sent the delete message
5. The protection SLAM validates and updates data base
6. ACK/NACK sent back to working SLAM. If a NACK is received, we fail the protection SP so that it will sync up again when it reboots.
   ~~Issue 9:~~Issue 4: If NACK, do we reboot protection SP. *Answer: yes*
7. Return to TL1 user the result of the command.

a. Send command to Working CLAM. No validation done at the CLAM
b. Send command to apps. Delete is validated and then deletion is completed.
c. If valid deletion, then send an asynchronous delete command to protection apps with a PSF call.
d. Working applications send ACK/NACK to CLAM.

e. CLAM sends ACK/NACK back to working SLAM

8.4 DLT Commands with Telica UnConfig Request



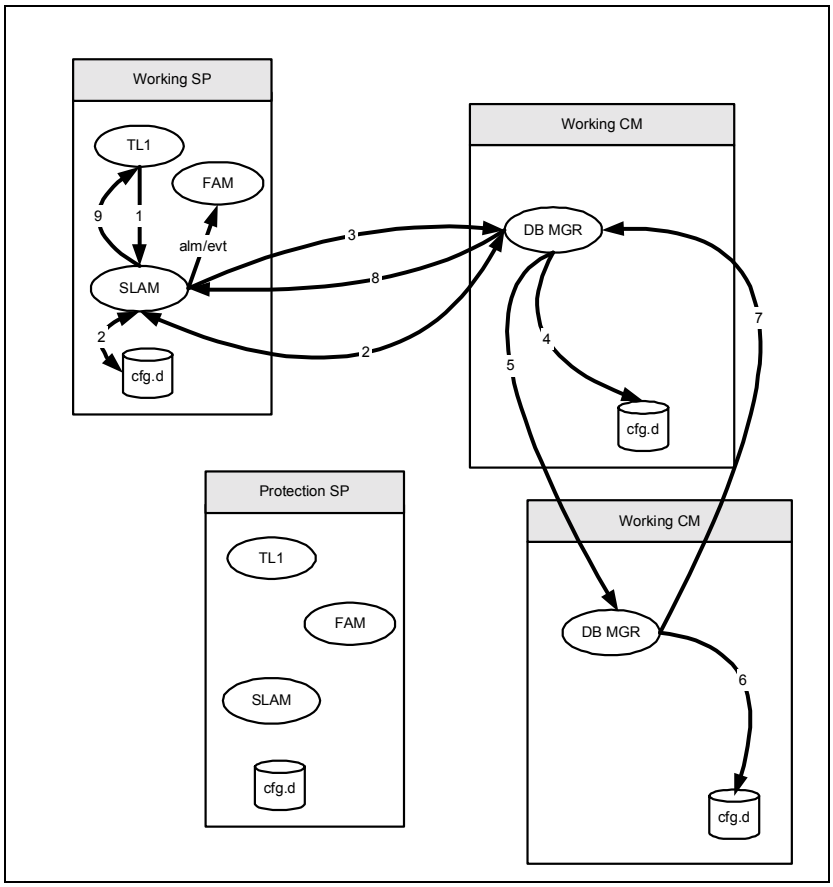
This method is used in Telica developed applications like CAS and MGCP. This sends a new type of config message called UNCONFIG. In this mode, the delete is done on the working side, and then the SLAM would forward it to the protection applications.

1. TL1 DLT command comes to SLAM
2. SLAM Validates provisioning and If an ACK, commit change to database. If NACK, send error response to TL1.
3. Send delete to Standby SP
4. Validate command and update disk
5. Send ACK/NACK to SLAM
6. Return to TL1 user the result of the command.

a. Send command to Working CLAM. No validation done at the CLAM
b. Send command to apps.
c. ACK/NACK sent back to CLAM
d. The UNCONFIG request sent to the protection CLAM. No validation is done.
e. Command forwarded to the applications where delete is done
f. ACK/NACK is sent back to the CLAM

g. ACK/NACK is sent back to the working CLAM
h. Result is sent back to SLAM

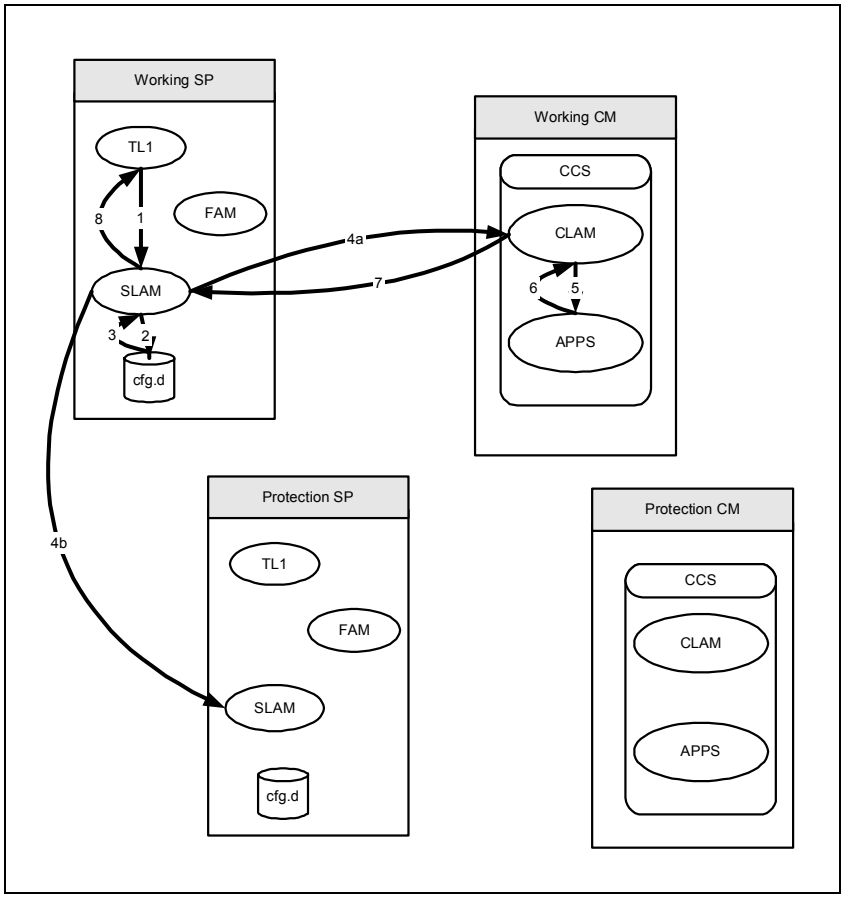
8.5 DLT Commands for Subscriber Applications



In this mode, the delete is done on the working side, then the DB Manager would forward it to the protection app.

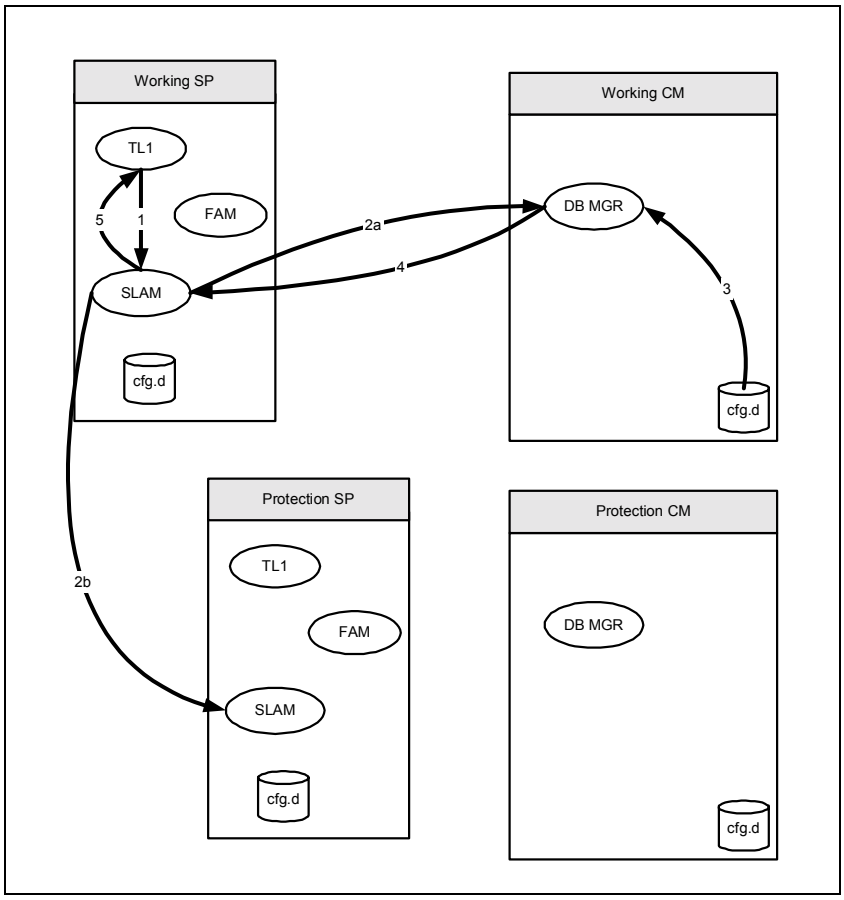
1. TL1 DLT command comes to SLAM
2. SLAM validates the command. It may need to read from local and remote datastores for the validation.
3. The SLAM sends a DB Write to the protection DM Manager.
4. The working DB manager will delete the record.
5. The working DB manager forwards the DB Write to the protection.
6. The protection record is deleted.
7. The ACK/NACK is sent back to working DB Manager
8. The response is sent back to the SLAM.
9. The response is sent to TL1

8.6 RTRV Commands



1. RTRV command is received at the working SLAM.
2. If there is provisioned data required, a data request is sent to timesTen
3. Data is returned
4. If there is operational state required for the RTRV, then:
   - 4a. the request is sent to the working CLAM. If the CLAM is not present, then the TL1 will have to return with an error that the resource is not available.
   - 4b. This arrow no longer applies and should be removed from diagram.
5. The data request is sent to the applications
6. The data is returned
7. The data is returned
8. The operational data is combined with the provisioned data and returned to the TL1.

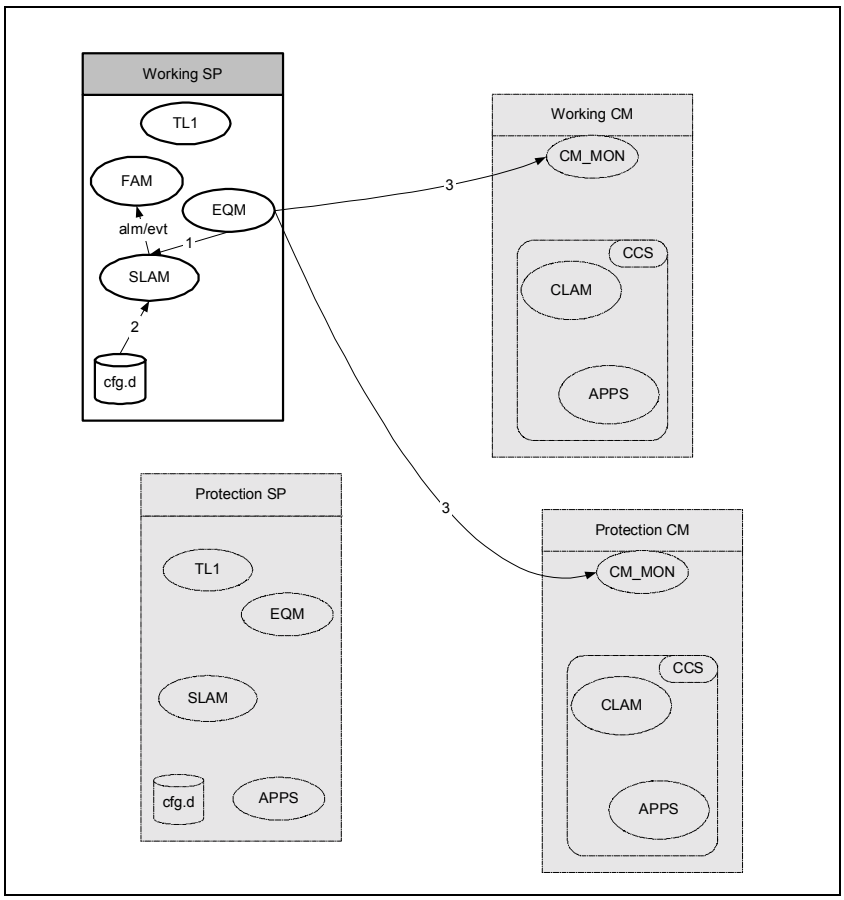
8.7 RTRV Commands (Subscriber)



1. RTRV command is received at the working SLAM.
2. If there are no CCM's running, then return to TL1 an error indicating that resources are not available.
   2a. The SLAM does a DB Read from the working DB Manager.
   2b. A transaction Id is sent to the protection SP SLAM for state information
3. Data is read from the DB data store
4. The data is returned to the SLAM
5. The data is returned to the TL1

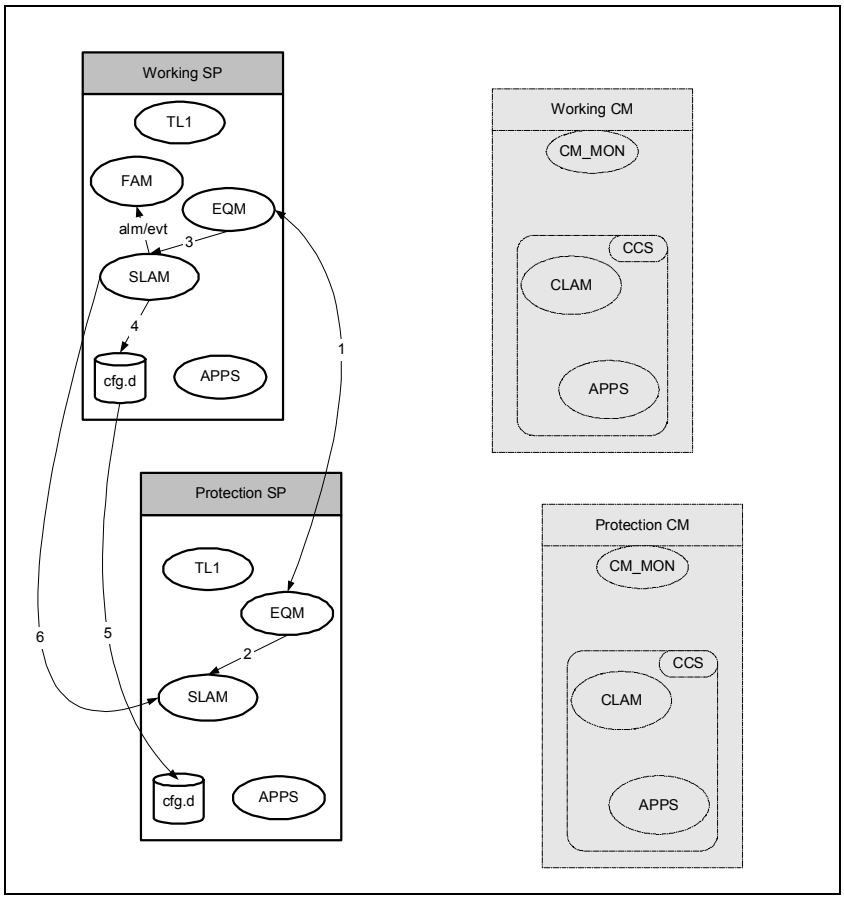
9 Board State Transitions

9.1 SLAM OOS→ ACT



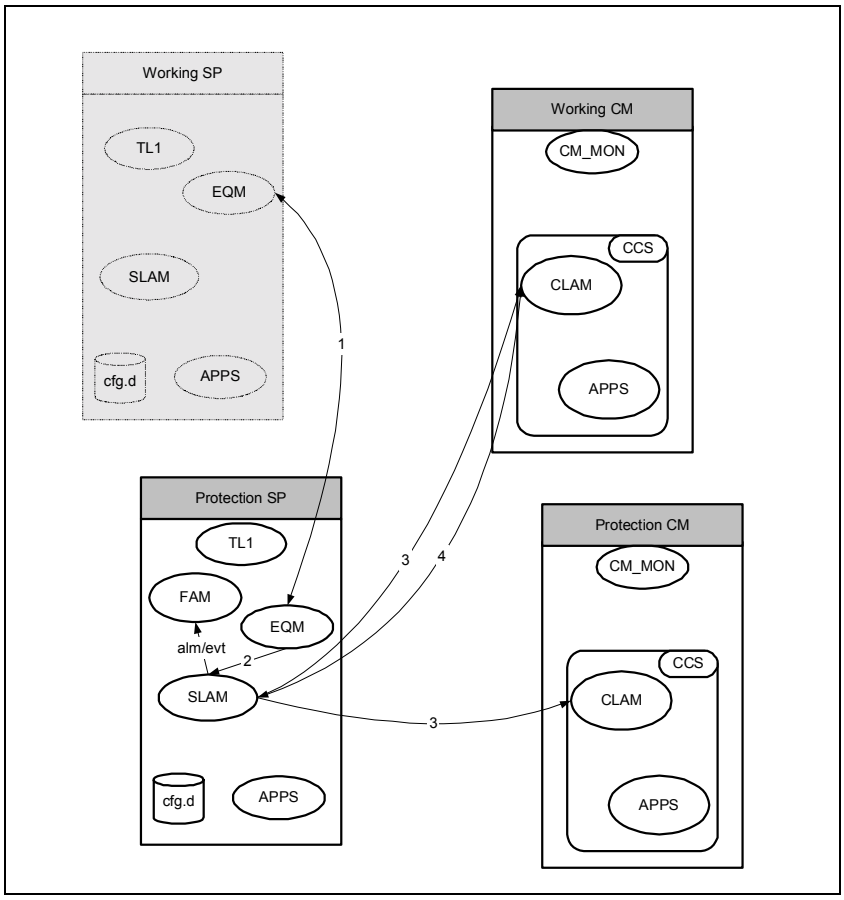
1. The Active EMF sends command to local SLAM indicating boot is complete and whether or not the SLAM is active.
2. SLAM will read the database. Not much to do here if no application resides on the SP that depends on the SLAM.
3. EQM will reboot all CMs (full board level reboot) to make sure they are up after the active SP. The first CM that boots up will be declared as the ACTIVE CM. Note, that means that the CM's actually will go through 2 reboots when a system is first powered up.

9.2 SLAM OOS→ STBY



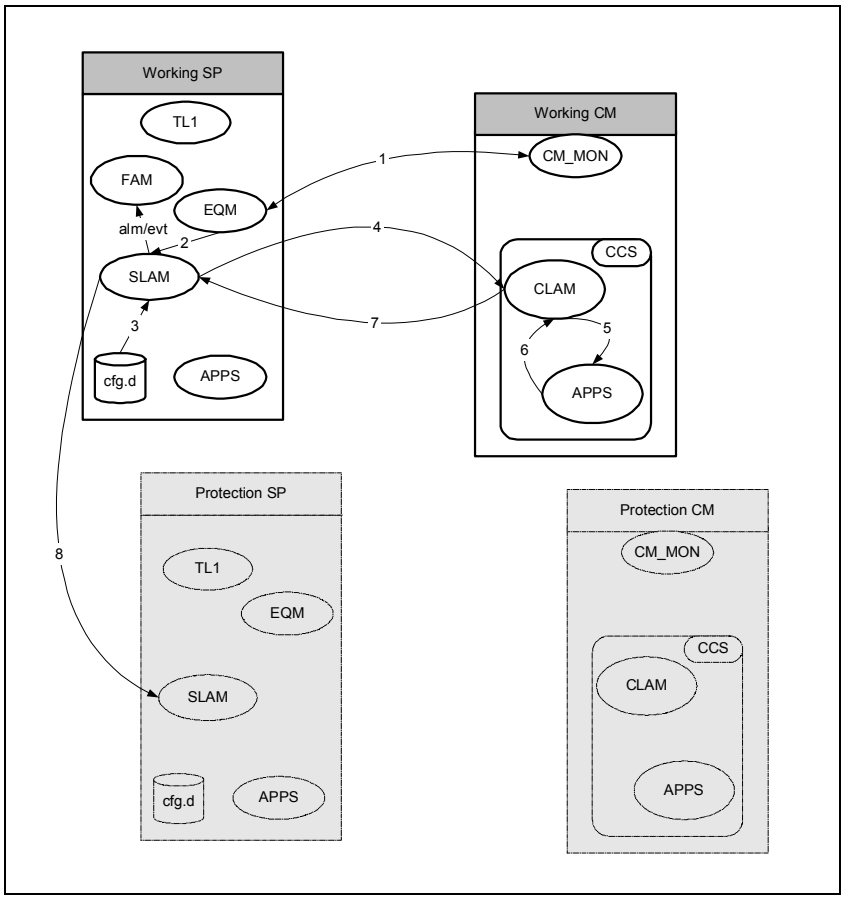
1. EMFs negotiate who is active.
2. Standby EQM indicates that local SLAM is the standby mode
3. Active EQM tells SLAM that it has a standby cold SLAM that needs replication
4. Active SLAM tells TimesTen to start replication
   ~~Issue 10:~~Issue 5: Do we want to use timesTen replication here or still go with the SLAM –SLAM replication that we are using for run time changes? *Use SLAM-SLAM replication*
5. Replication is done.
6. Dynamic SLAM data is replicated to standby SLAM (warm start)

9.3 SLAM STBY→ACTIVE



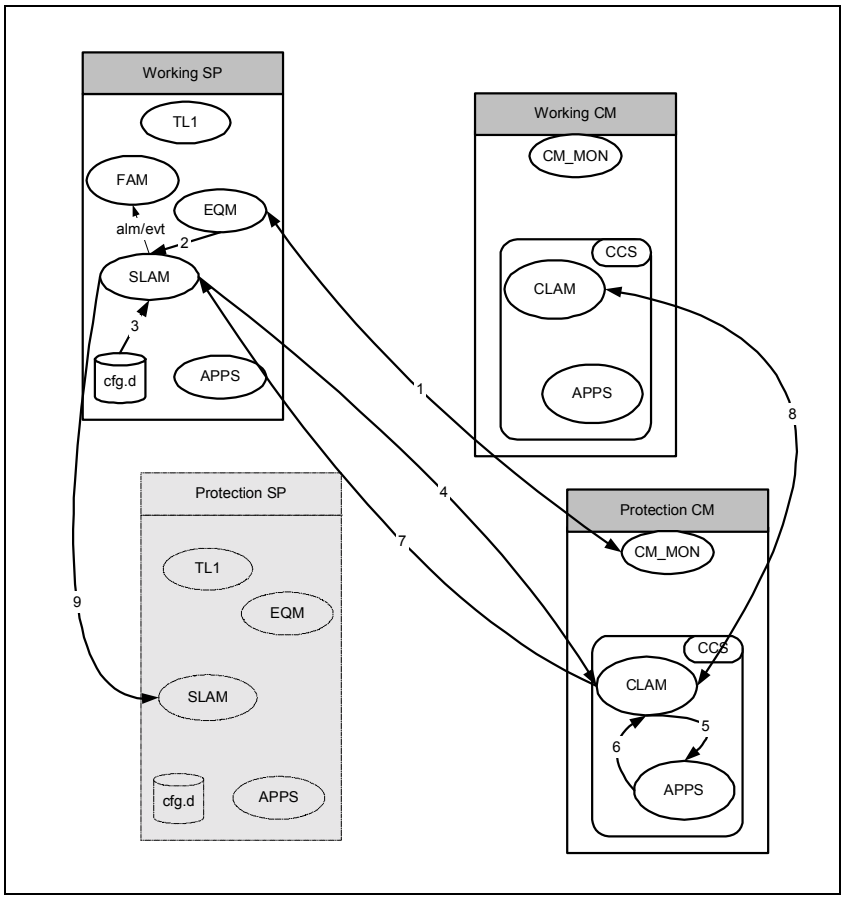
1. EMFs communicate and protection SP is made active.
2. EMF indicates that this board is now active
3. SLAM broadcasts that board is now active to all CMs
4. Audit between SLAM and CLAMs. This is needed since there may be pending configurations that did not make it to the CLAM or the response did not make it back. The other issue could be that an alarm or event was lost.

9.4 CLAM OOS→ ACT



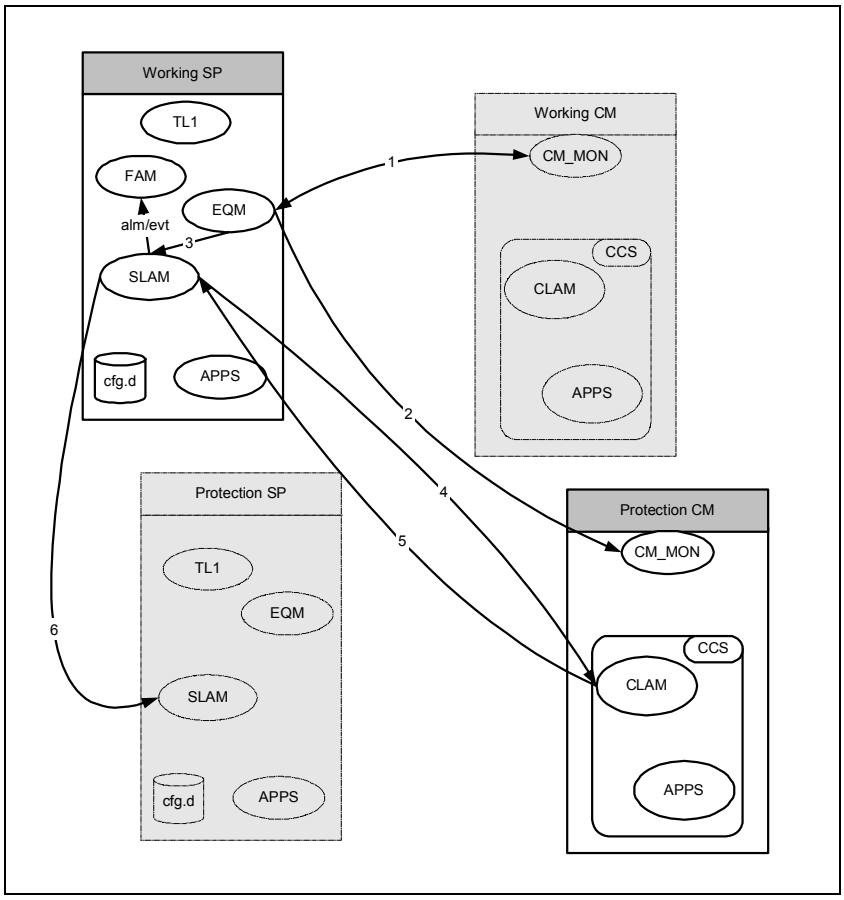
1. EMF and CM_MON communicate when CM is operational.
2. EMF tells SLAM that a working CM is IS
3. SLAM will read the database
4. SLAM sends provisioning data to CLAM
5. CLAM configures applications
6. ACK/NACK is received at CLAM
7. ACK/NACK is received at SLAM
8. If protection SP is present, then send down the new CM information.

9.5 CLAM OOS→ STBY



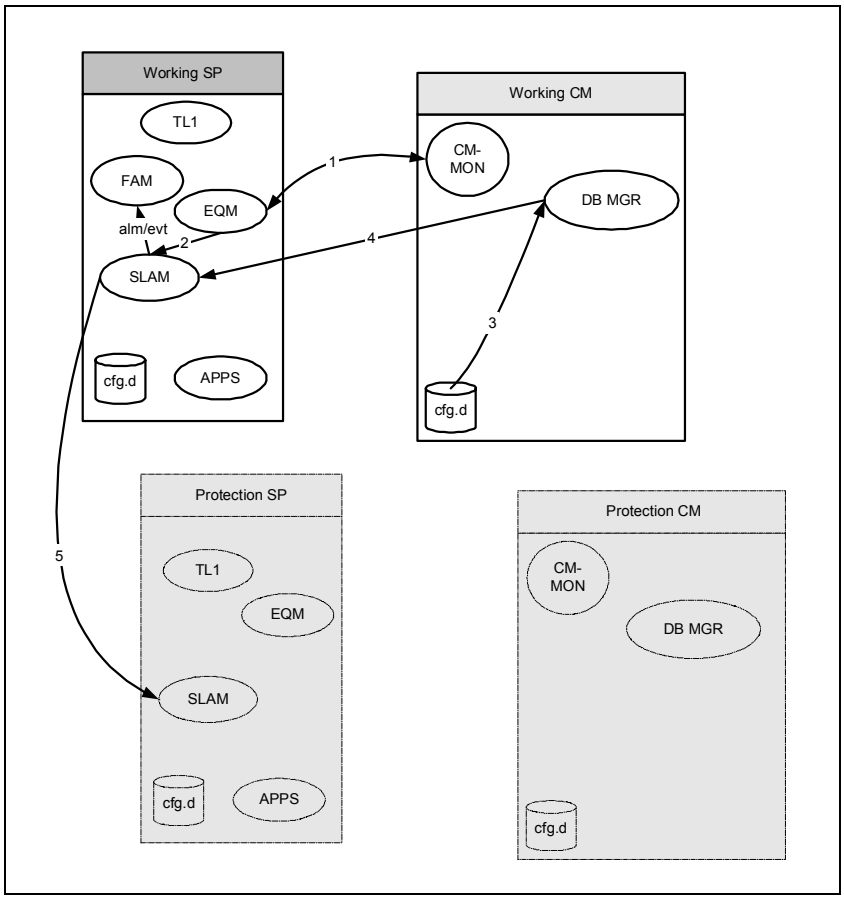
1. EMF and CM_MON communicate when CM is operational.
2. EMF tells SLAM that a protection CM is IS
3. SLAM will read the database
4. SLAM sends provisioning data to CLAM
5. CLAM configures applications
6. ACK/NACK is received at CLAM
7. ACK/NACK is received at SLAM
8. Working and protection CLAM setup communication path to enable dynamic replication.
9. If protection SP is present, then send down the new CM information.

9.6 CLAM STBY→ ACTIVE



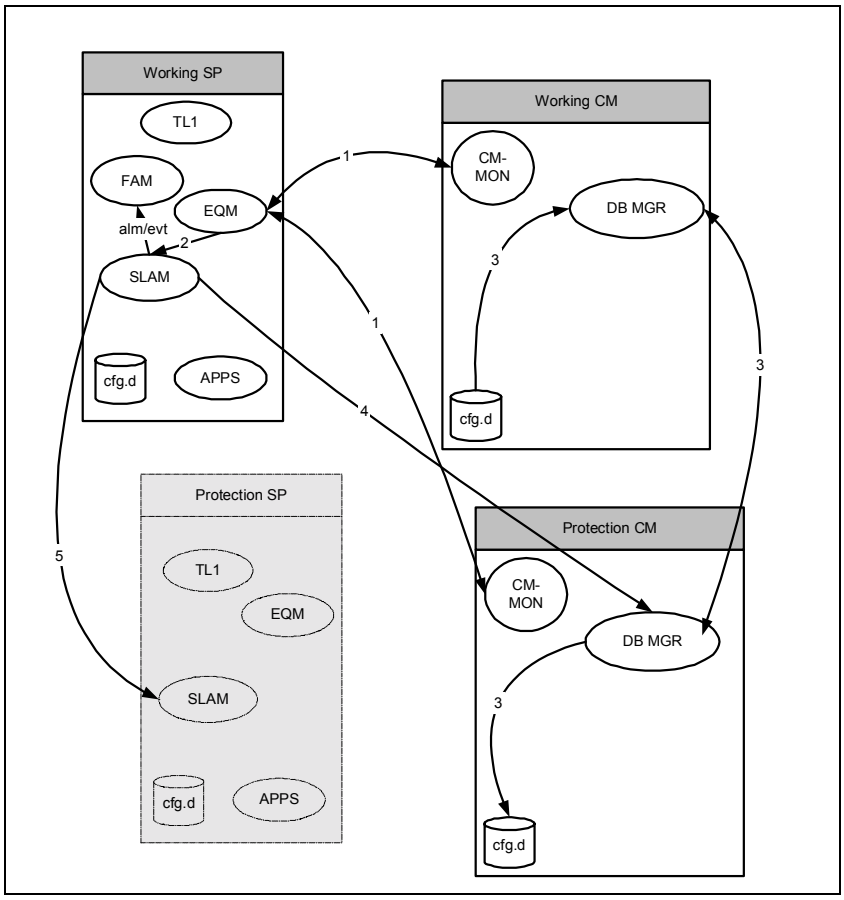
1. EMF on working SP detects that working CM is down.
2. EMF tells protection CM that it is now working
3. EMF tells SLAM that there is a new working CM
4. SLAM sends command to old protection CLAM that it is now the working unit
5. CLAM acknowledges back
6. If protection SP is present, then send down the new CM information.

9.7 DB Manager OOS→ ACT (Subscriber)



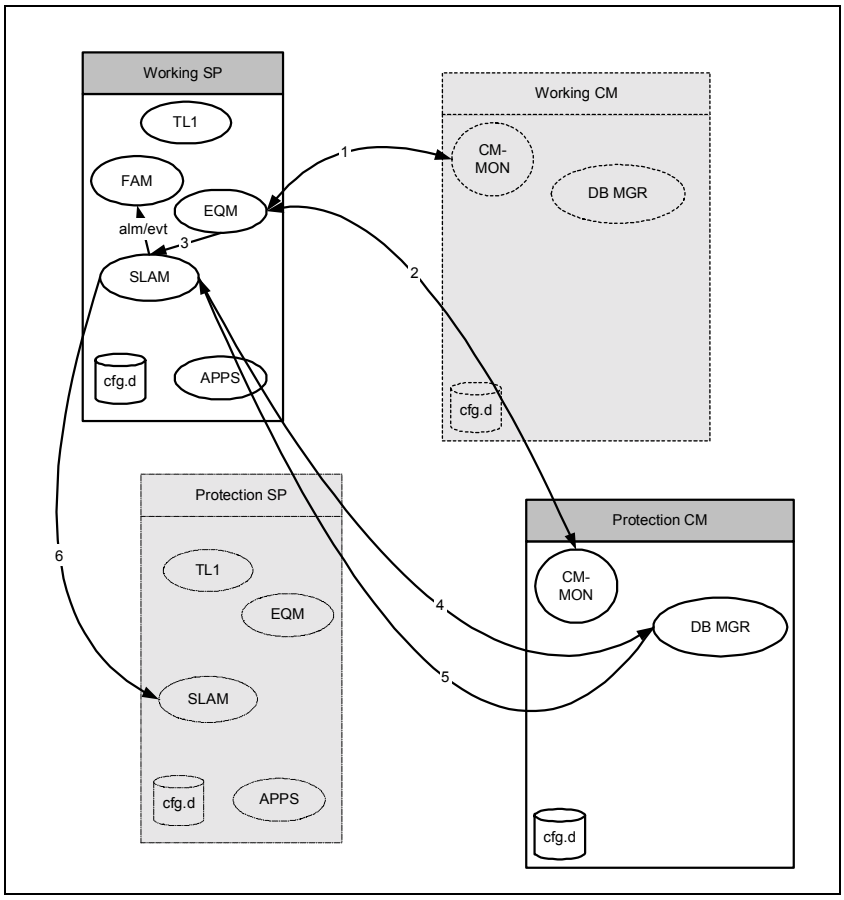
1. The working CCM comes up and communicates with working SP.
2. EQM tells SLAM that there is a working CCM.
3. DB Manager is initialized and ready to accept dB queries
4. Status message sent to SLAM to indicate ready (with no stby)
5. Protection SP is updated with state of CM.

9.8 DB Manager OOS→ STBY (Subscriber)



1. The standby CM comes up and communicates with working SP and working CCM.
2. EQM tells SLAM that there is a protection CCM.
3. DB Manager replicates data to standby
4. Status sent to SLAM as Standby Hot
5. Update protection SLAM with new CCM status

9.9 DB Manager STBY→ ACTIVE (Subscriber)



This is the same steps as in normal DB Manager STBY→ACTIVE.

1. EMF on working SP detects that working CCM is down.
2. EMF tells protection CCM that it is now working
3. EMF tells SLAM that there is a new working CCM
4. SLAM sends command to old protection DB Manager that it is now the working unit. Also sends message to all XU in system.
5. DB Manager acknowledges back
6. If protection SP is present, then send down the new CM information.

10 SLAM DESIGN

10.1 Process, Relay and Trillium ProcIds

The SLAM process will communicate with all the CCS suites and other SLAMs using relay.

10.2 New Macros for procId management for 5.0

These macros will be located in TelicaRoot/components/signaling/sigcom/telica_pid.h

- Macro to get partner CM slot
- Macro to get procId of a CCS process (CLAM, SIP etc), takes in (CM slot, CPU, CCS suite Id)
- Macro to get partner CCS procId "set"

Note: ProcId is defined in 1) 79-5017, Media Gateway Controller (MGC) Architecture Specification

10.3 Relay clients and servers

SLAM will initialize relay and setup relay server sockets to accept connections from CLAMs (CCSs). SLAM will also initialize the relay channel to the SLAM on the CCM.

10.4 SLAM initialization

This initialization is based on how the current signaling process starts up.

1. EMF forks the SLAM process
2. `main()` in the SLAM process:
   - does any process priority adjustment by calling `setpriority()`,
   - sets up arguments for procInit required by `gitGAInit()`,
   - initializes the GA interface, `gitGAInit()`, `gitGAInit()`:
     - opens the HAPI endpoint
     - starts EMF heartbeating on the GA interface
     - registers events that we'll be publishing
     - subscribes to events that we'll be receiving
   - initializes other global structures used the GA task and MTSS

   On the **ACTIVE**:
   - polls the GA interface for svcmgr to startup

   ~~opens the Times10 ODBC connection~~

   On the **STANDBY**
   - polls the GA interface for svcmgr to startup~~sequences control with Times10 bulk replication~~

   On **ACTIVE/STANDBY**
   - starts the MTSS initialization function

- MTSS initialization:
  - Registers activate-init and activate-task functions and starts the SLAM, GIT, relay, FTHA (message router, system agent, system manager), watchdog threads, and any other required threads.

3. The activate-init for SLAM, `slmActvInit()`:
   - sets up some global datastructures, state of the SLAM config FSM
   - setup/init CM and CCS FSMs
   - Start the config FSM –
4. The SLAM config FSM does
   - genCfg relay, setup relay channels, setup SLAM FTHA
   - wait for DBP CLAM to come into service and preform~~any~~ SLAM specific DB rewind
   - Confirm back to EMF – "process ready"
5. On ACTIVE: do equipment audit, wait for standby available, start syncing, confirm back to EMF
6. On STANDBY: wait for STANDBY_COLD state, procInfo stuff
7. On failover, do another equipment audit

~~Some logic will be reordered depending on how we decide to do DB access. We may have to do additional DB initialization for the non-blocking Ties10 API model~~.

10.5 CM Management

SLAM keeps a control block for each CM in an array. This control block is initialized after ent-eqpt is executed for a particular CM.

10.5.1 Data Structures

CM control block:

```
typedef struct slCmCb
{
    U8       slot;
    U8       state;
    char     clei[CLEI_MAX_CHAR];
    U8       swVersion[SW_VERSION_LEN+1];
    SlCcsCb  *ccsCb;  /* pointer to CCS array */
    /* ... */
} SlCmCb;
```

An array of type `SlCmCb` will hold information for all CM modules. `clei[]` and `swVersion[]` are set when SLAM gets the BRD_CREATED event from EQM. An array of SlCcsCb will hold information for all CCSs in that CM.

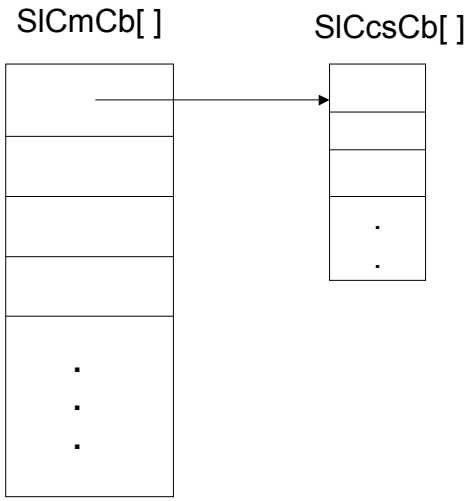


Figure 10-1

Notes: This datastructure can change - we may also want to have a CPU hierarchy available.

10.5.2 Logical FSM instance organization

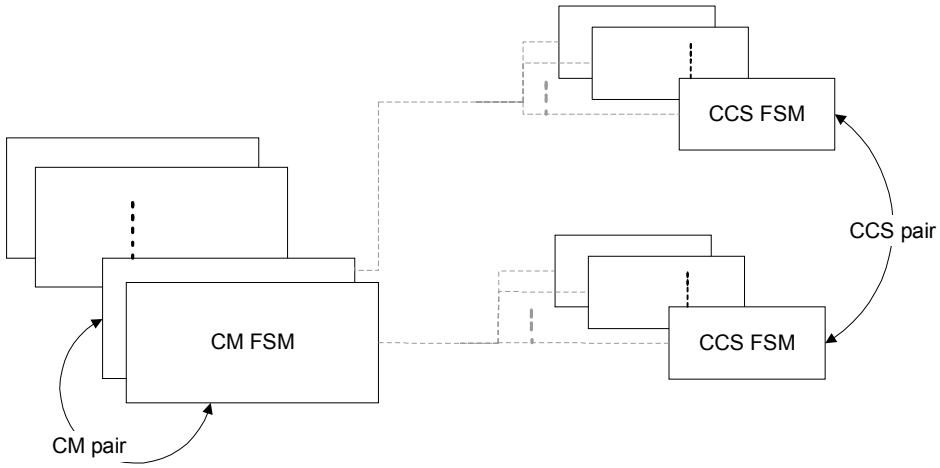


Figure 10-2

10.5.3 CM States

The states used for the CM FSM are located at:
Telica/components/signaling/lm/slam/base/sl_brd.h

```
enum
{
        SLBRD_ST_OOS = 0,
        SLBRD_ST_ACT_STANDALONE,
        SLBRD_ST_ACT_SYNCED,
```

```
        SLBRD_ST_SBY_NOTSYNCED,
        SLBRD_ST_SBY_SYNCED,
    CM_OOS = 0,             /* CM OOS, waiting for EQM to tell us IS */
    CM_ACT_IS,              /* CM is active IS, CCSs may or may not be IS */
    CM_SBY_IS,              /* CM is standby IS, CCSs may or may not be IS */
    CM_MAX_STATE
};
```

Note that there are not many states to track CM. Most process syncing, synced states are CCS specific.

10.5.4 CM FSM Events

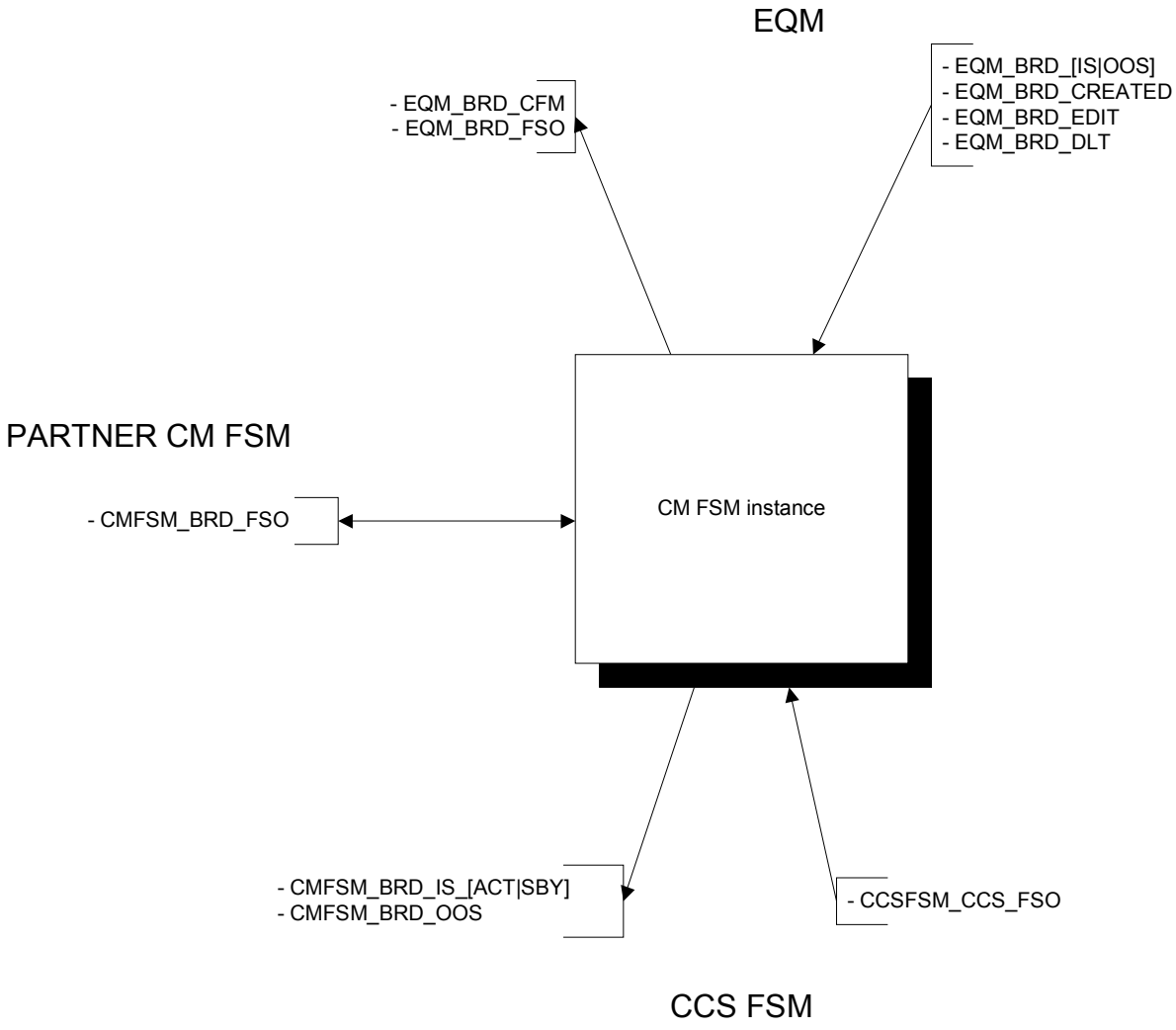


Figure 10-3

10.5.5 CM State Machine

10.5.5.1 SDL Representation Guide

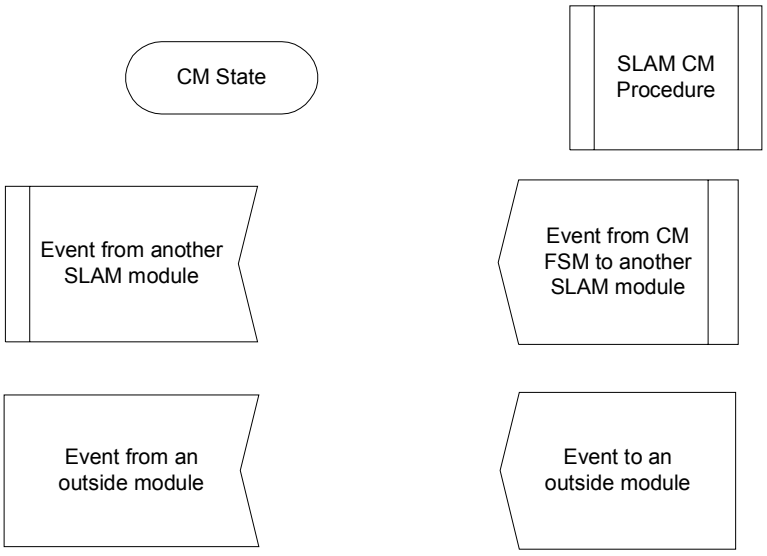


Figure 10-4

10.5.5.2 Common states

10.5.5.2.1 CM_OOS

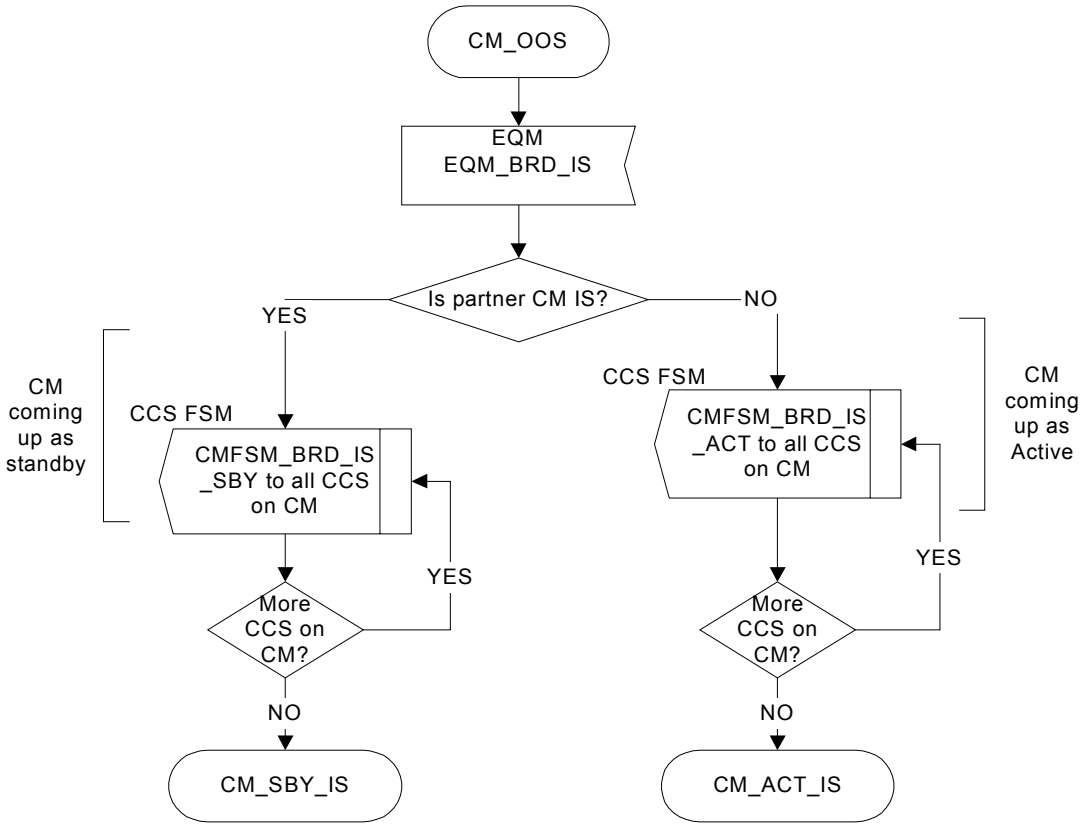

CM_OOS
EQM
EQM_BRD_IS
Is partner CM IS?
YES
NO
CM coming up as standby
CCS FSM
CMFSM_BRD_IS _SBY to all CCS on CM
YES
More CCS on CM?
NO
CM_SBY_IS
CCS FSM
CMFSM_BRD_IS _ACT to all CCS on CM
CM coming up as Active
YES
More CCS on CM?
NO
CM_ACT_IS

Figure 10-5

10.5.6 Active state machine

10.5.6.1 CM_ACT_IS state

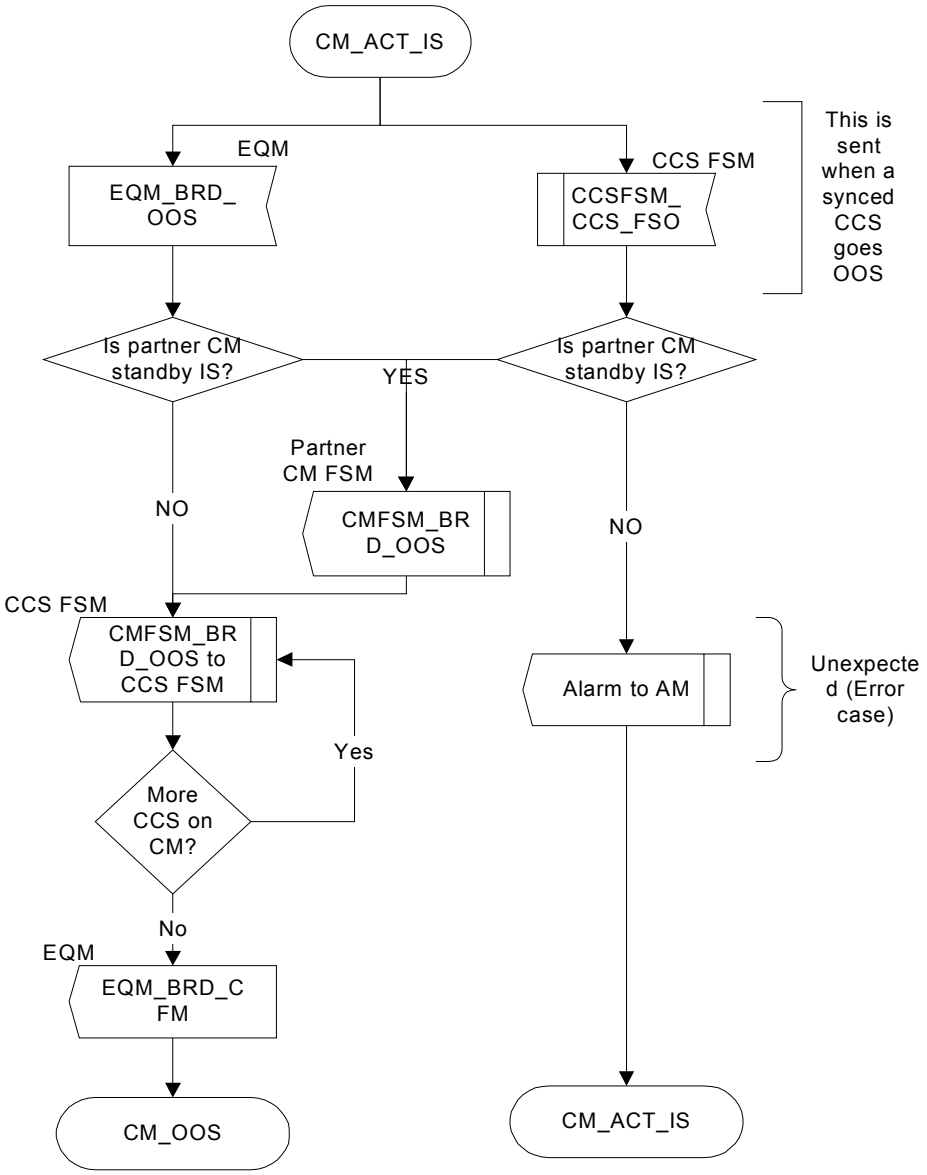


Figure 10-6

10.5.7 Standby state machine

10.5.7.1 CM_SBY_IS state

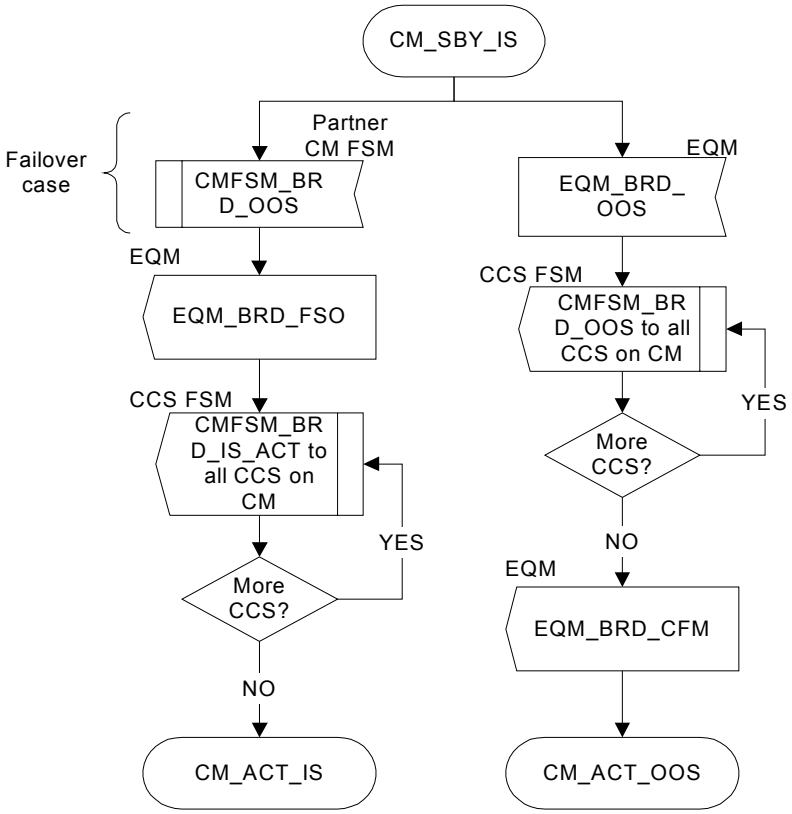


Figure 10-7

10.6 CCS Management

10.6.1 Data Structures

There is a CCS control block per CCS. The CM control block has a pointer to an array of CCS control blocks (Figure 10-1). Note that CCS refers to the CCS suite of processes on that CM. This structure can be found at
Telica/components/signaling/lm/slam/base/sl_brd.h

```
typedef struct slCcsCb
{
    U8   slot;         /* back reference to CM slot */
    U8   state;
    U8   cpu;          /* CPU that this CCS is running on */
    U8   relayState;   /* TRUE:up, FALSE:down */
    U8   clamState;    /* TRUE if CLAM notified IS to SLAM, FALSE otherwise */
```

```
    U8  mgName[MAX_MG_NAME];  /* this would be all '\0' if no MG associated */
    U32 numCps;       /* used for CCS assignment */
    U32 rxSeqNum;     /* tracks sequence numbers for outgoing messages */
    U32 txSeqNum;     /* tracks sequence numbers for incoming messages */
    /* other stuff .. */
} SlCcsCb;
```

A CCS suite hosting an MGC will have a valid mgName filled in.

`clamState` is TRUE if the CLAM started up in the CCS suite, and sent the initial IS event to the SLAM. The CCS has no configuration at this point. This state is reset when the CCS goes OOS.

10.6.2 CCS selection

A CCS suite is selected automatically when ent-mg-assoc is executed. The CCS selection algorithm is defined in [2].

Notes: future development: we may want to do runtime allocation of MGs to CCSs. Example: If somebody pulls out a simplex CM (and if this is the only CM in the chassis for simplicity) because of a suspected problem in the slot/backplane, and re-inserts this CM into another simplex slot, we may want to move the CCS personalities from the previous slot to this slot automatically (after a timeout)? Also support TL1 commands to move CCSs.

10.6.3 CCS states

```
enum
{
    CCS_OOS=0,
    CCS_IDLE,
    CCS_BOOTING_ACT,
    CCS_BOOTING_STBY,
    CCS_ACTIVE_STANDALONE,
    CCS_ACTIVE_SYNCING,
    CCS_ACTIVE_SYNCED,
    CCS_STANDBY_READY,
    CCS_STANDBY_SYNCING,
    CCS_STANDBY_HOT,
    CCS_MAX_STATE
};
```

These states are similar to the existing EQM states for CPU in 3.8. Note that there is a `CCS_OOS` state and a `CCS_IDLE` state. These were not required in 3.8. `CCS_STANDBY_COLD` has been renamed to `CCS_STANDBY_READY`. This state indicates that the standby CCS is booted up and configured, and is ready to accept warmstart and runtime updates from the active CCS. `CCS_STANDBY_SYNCING` is a new state to indicate that standby is syncing.

A CCS is in the OOS state when either the CM hosting the CCS is OOS, or when the SLAM cannot communicate with the CCS (relay not up, or CLAM on the CCS has not sent the initial IS event).

A CCS is in the CCS_IDLE state when the SLAM can communicate with the CCS process suite, but no MG association has been assigned to the CCS. The CCS_OOS and CC_IDLE states can exist for a CCS on both the active and the standby CM.

A CCS is in the CCS_BOOTING_ACT or CCS_BOOTING_STBY state when the configuration is being downloaded and playedback into the layers by the CLAM.

10.6.4 CCS FSM events

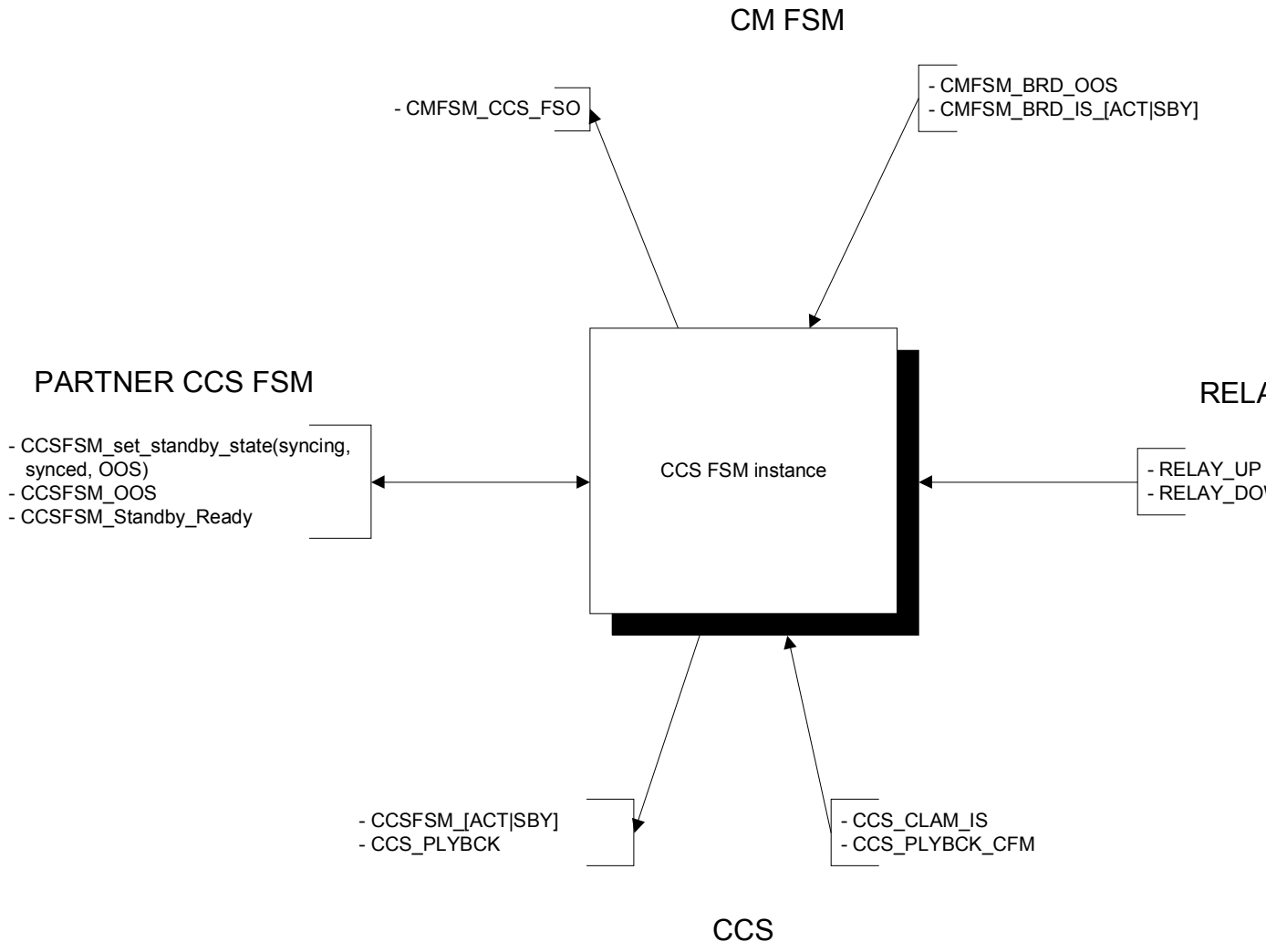


Figure 10-8

10.6.5 CCS State Machines

Major states are represented. There are minor states in the state machine, which are handled within procedures.

10.6.6 SDL Representation Guide

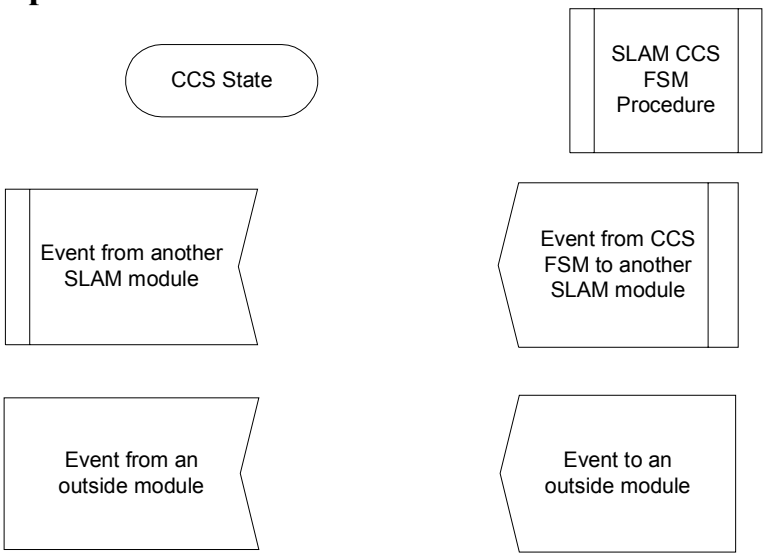


Figure 10-9

10.6.6.1 Common states

10.6.6.1.1 CCS_OOS

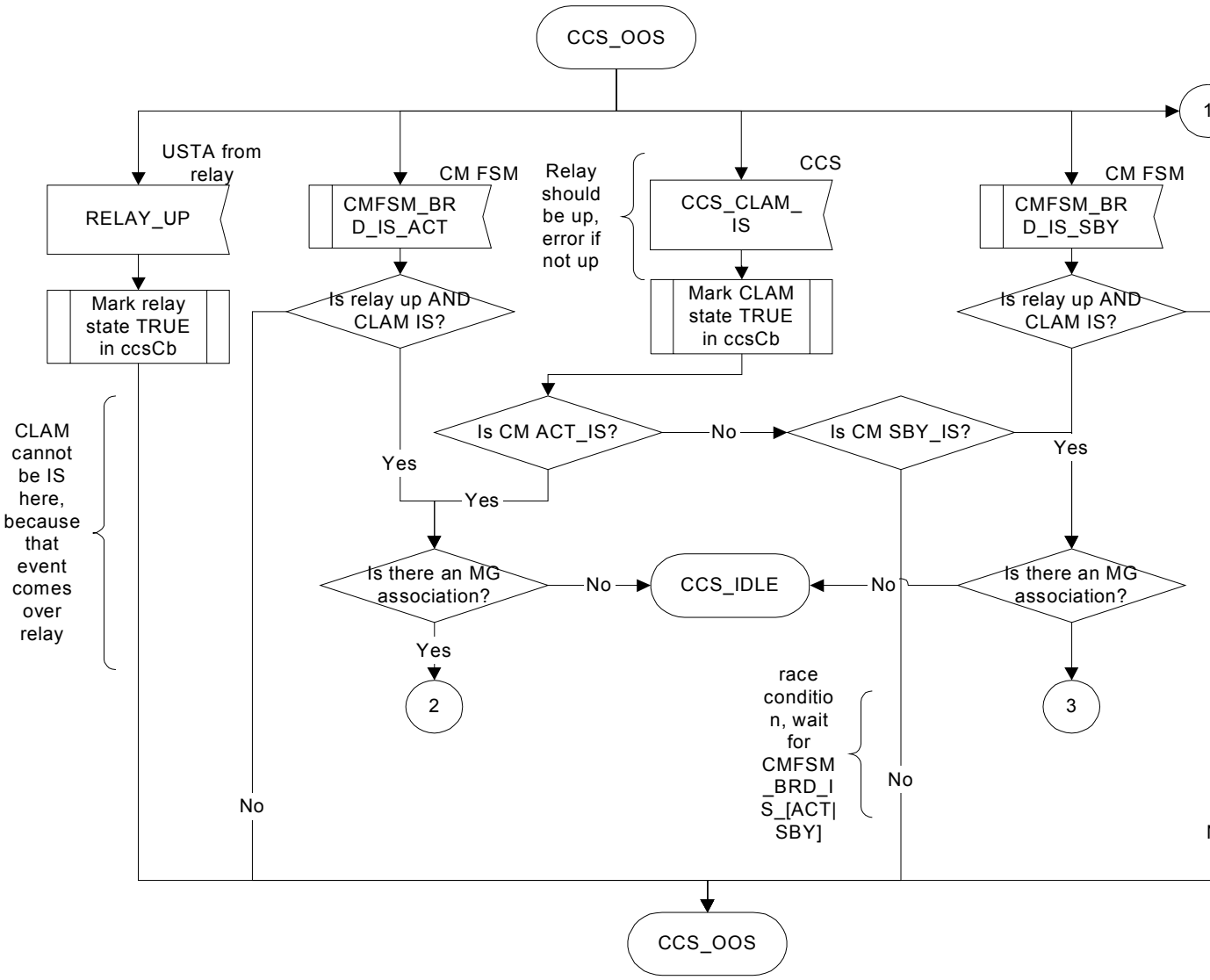

CCS_OOS
USTA from relay
RELAY_UP
Mark relay state TRUE in ccsCb
CM FSM
CMFSM_BRD_IS_ACT
Is relay up AND CLAM IS?
Relay should be up, error if not up
CCS
CCS_CLAM_IS
Mark CLAM state TRUE in ccsCb
CM FSM
CMFSM_BRD_IS_SBY
Is relay up AND CLAM IS?
CLAM cannot be IS here, because that event comes over relay
Is CM ACT_IS?
No
Is CM SBY_IS?
Yes
Yes
Yes
Is there an MG association?
No
CCS_IDLE
No
Is there an MG association?
Yes
2
race condition, wait for CMFSM_BRD_IS_[ACT|SBY]
No
3
No
CCS_OOS

Figure 10-10

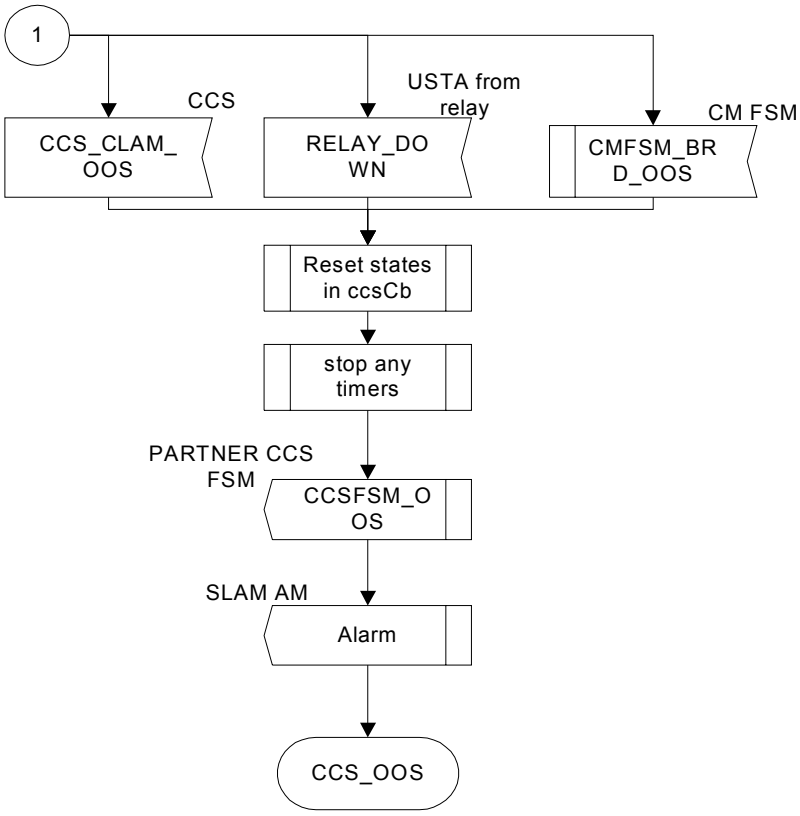

1
CCS
USTA from relay
CM FSM
CCS_CLAM_ OOS
RELAY_DO WN
CMFSM_BR D_OOS
Reset states in ccsCb
stop any timers
PARTNER CCS FSM
CCSFSM_O OS
SLAM AM
Alarm
CCS_OOS

Figure 10-11

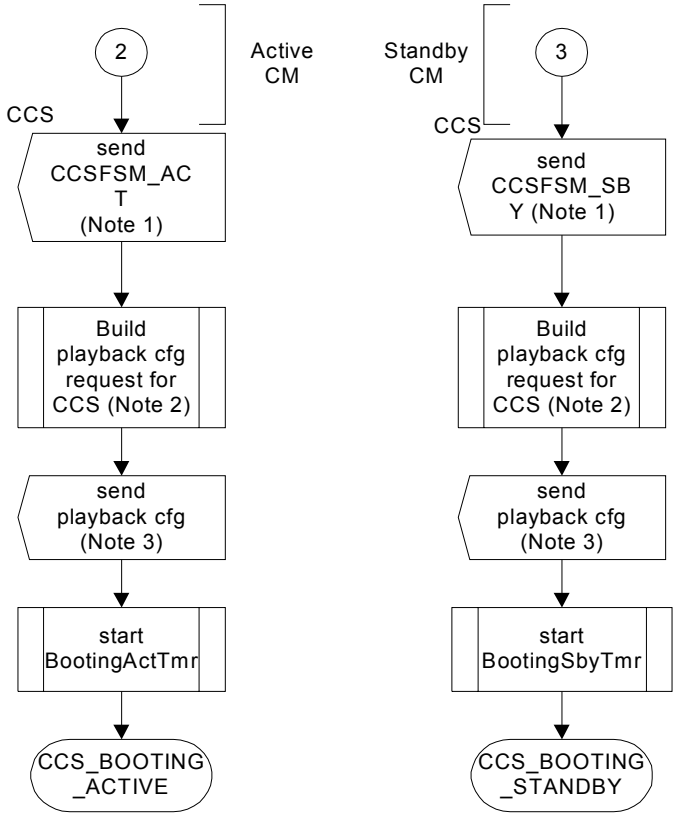


Figure 10-12

Notes:

1) This event can include mgName and other information required for initial configuration.
2) This procedure can be loosely coupled with the database to avoid SLAM thread lockups. This procedure uses the generic library for loading database rows into the mBuf~~, described in section ??.~~.
3) This will use the generic function to send segmented data to the CLAM and can be loosely coupled to enable flow control~~. This is covered in section ??.~~.

10.6.6.1.2 CCS_IDLE state

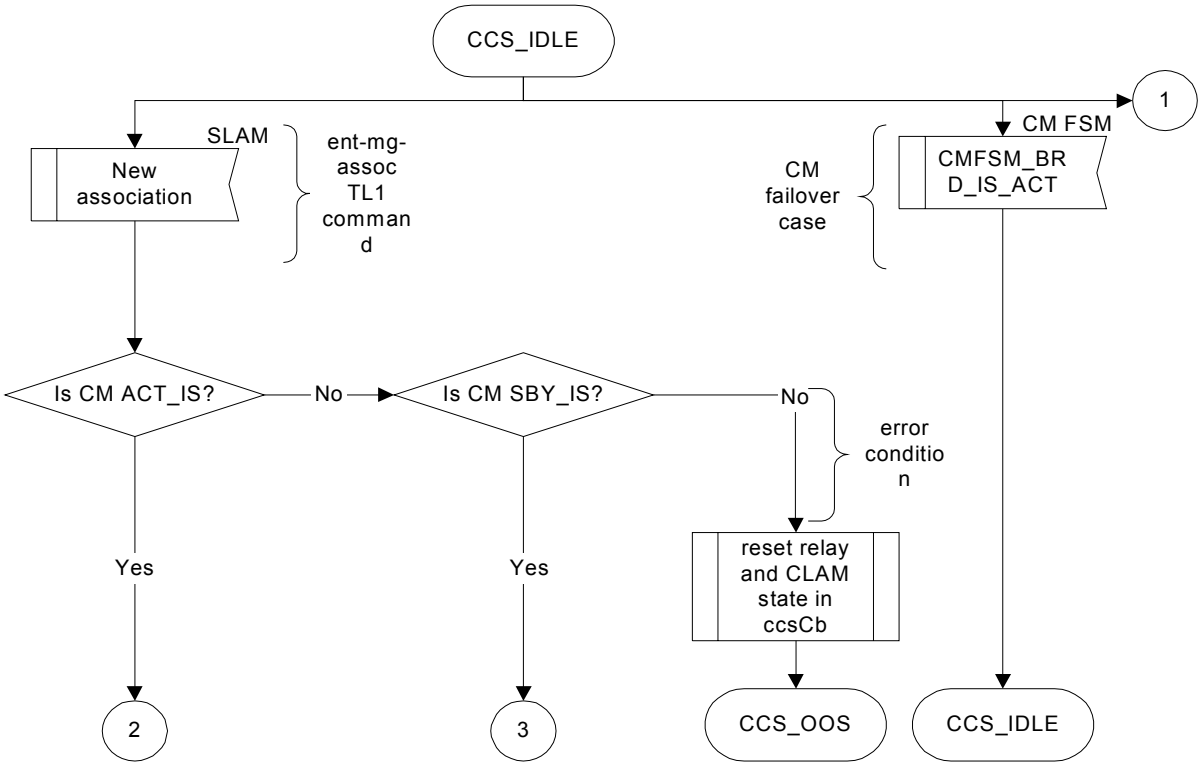


Figure 10-13

10.6.6.2 Active CCS state machine

10.6.6.2.1 CCS_BOOTING_ACT state

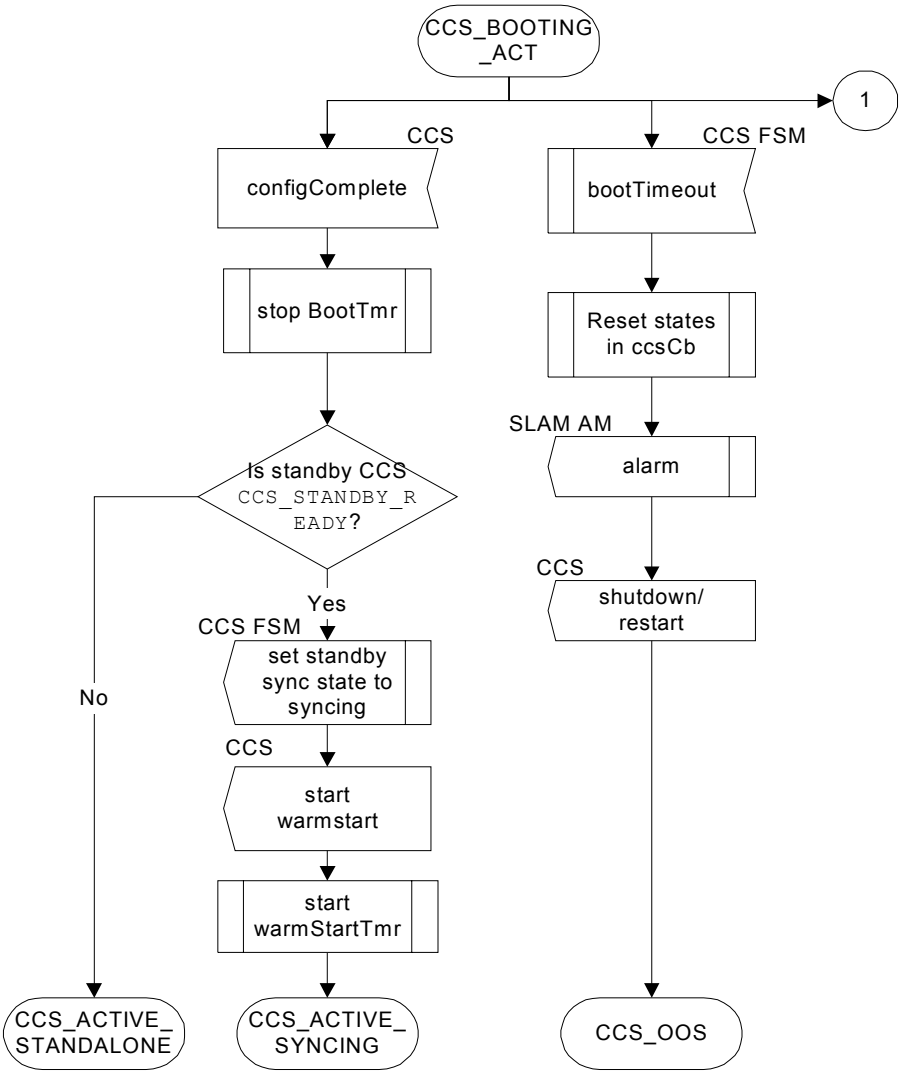


Figure 10-14

Note that the standby CCS can come up early before the active CCS. The standby will bootup in this case and get configured and wait for the active CCS to start the sync.

10.6.6.2.2 CCS_ACTIVE_STANDALONE state

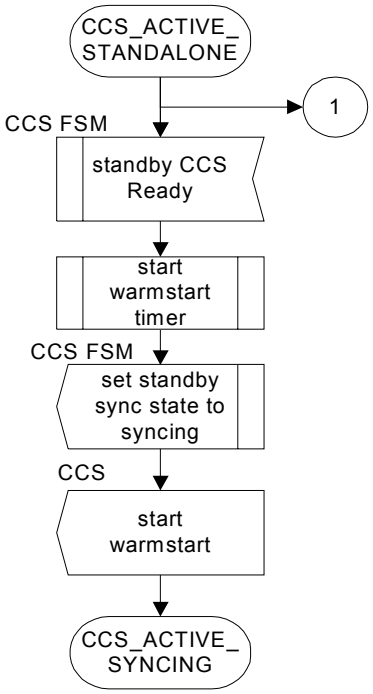


Figure 10-15

10.6.6.2.3 CCS_ACTIVE_SYNCING state

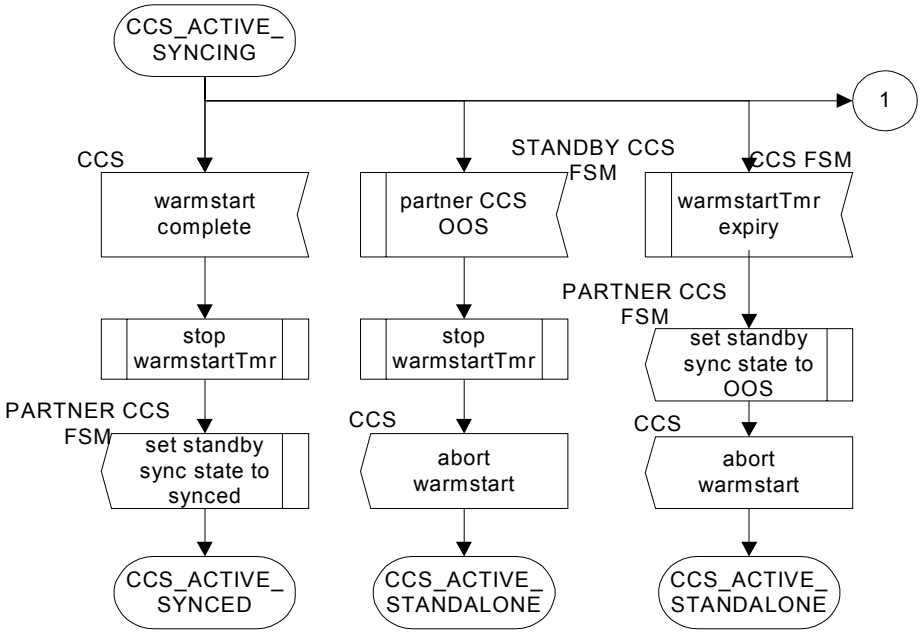


Figure 10-16

10.6.6.2.4 CCS_ACTIVE_SYNCED state

10.6.6.3 Standby CCS state machine

10.6.6.3.1 CCS_BOOTING_STBY state

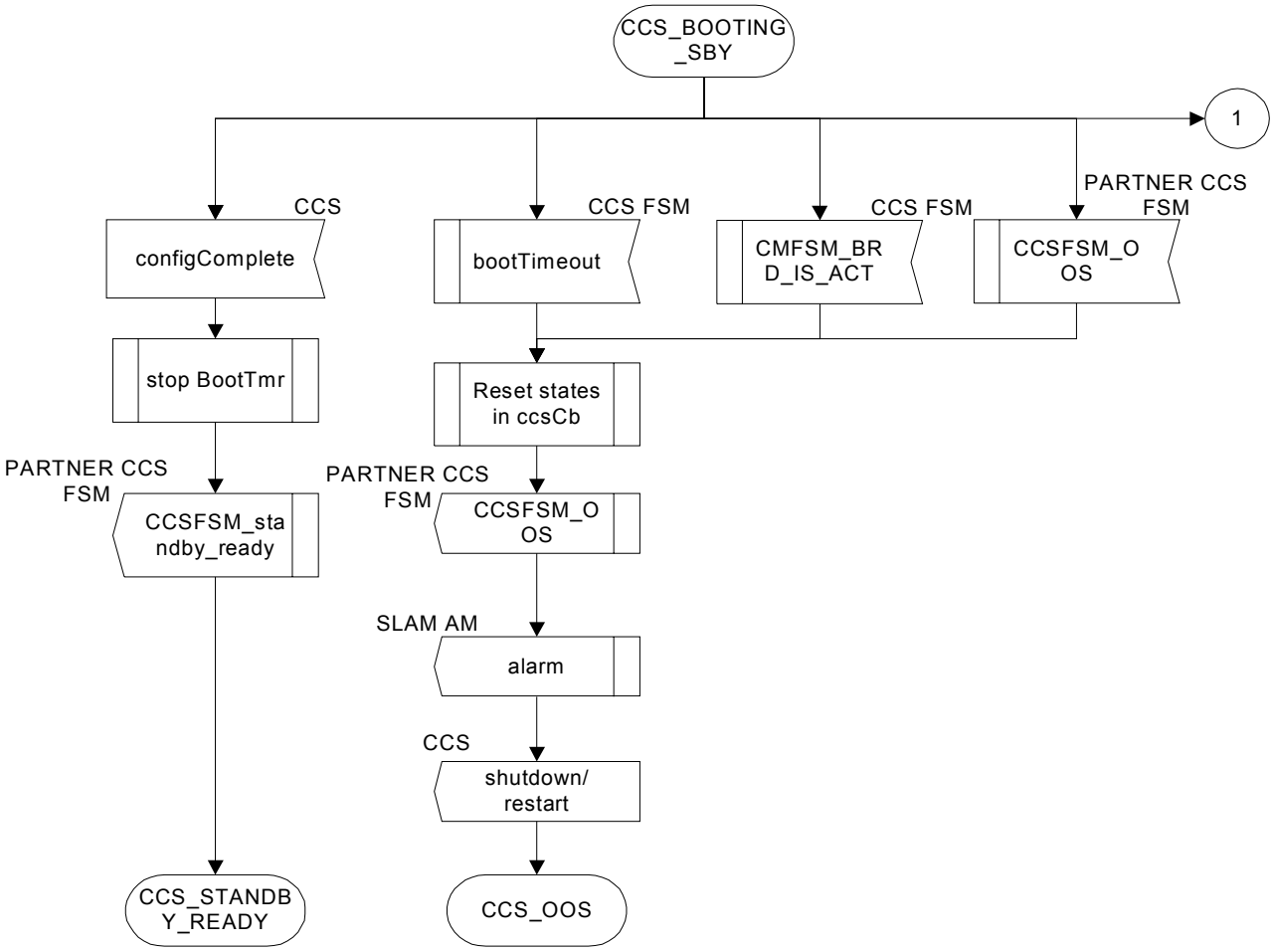


Figure 10-17

10.6.6.3.2 CCS_STANDBY_SYNCING state

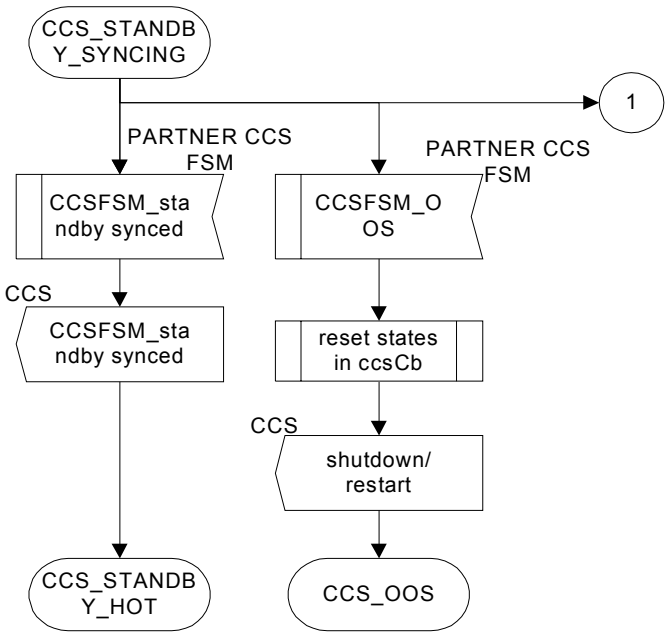


Figure 10-18

10.7 CM Failover

The SDLs in section 10.5 and section 10.6 describe the processing for CM Failover in the SLAM FSMs. This section details the flow control for failovers.

10.7.1 CCS FSM directed Failover

When a CCS FSM detects the CCS going OOS, and if the CCS was active synced, it will send a message to the EMF that may start a failover.

~~**Issue 11:**~~**Issue 6:** How do we handle the case when there are x number of MG's associated which means the CCS's are up and synchronized. Then a new MG is added, which starts the GenCfg of the MGI layer. This CCS is not synced yet. Then the active fails. We want to fail over to the standby, but the one CCS is not in sync. Do we tell the EMF that we are not in sync for a short period, or leave it in the sync state and then after the fail over, reset the one CCS suite? *Answer: The SLAM will tell the EMF that the active CCS failed and the EMF will fail over. The CCS that was coming up is restarted on the new active CM.*

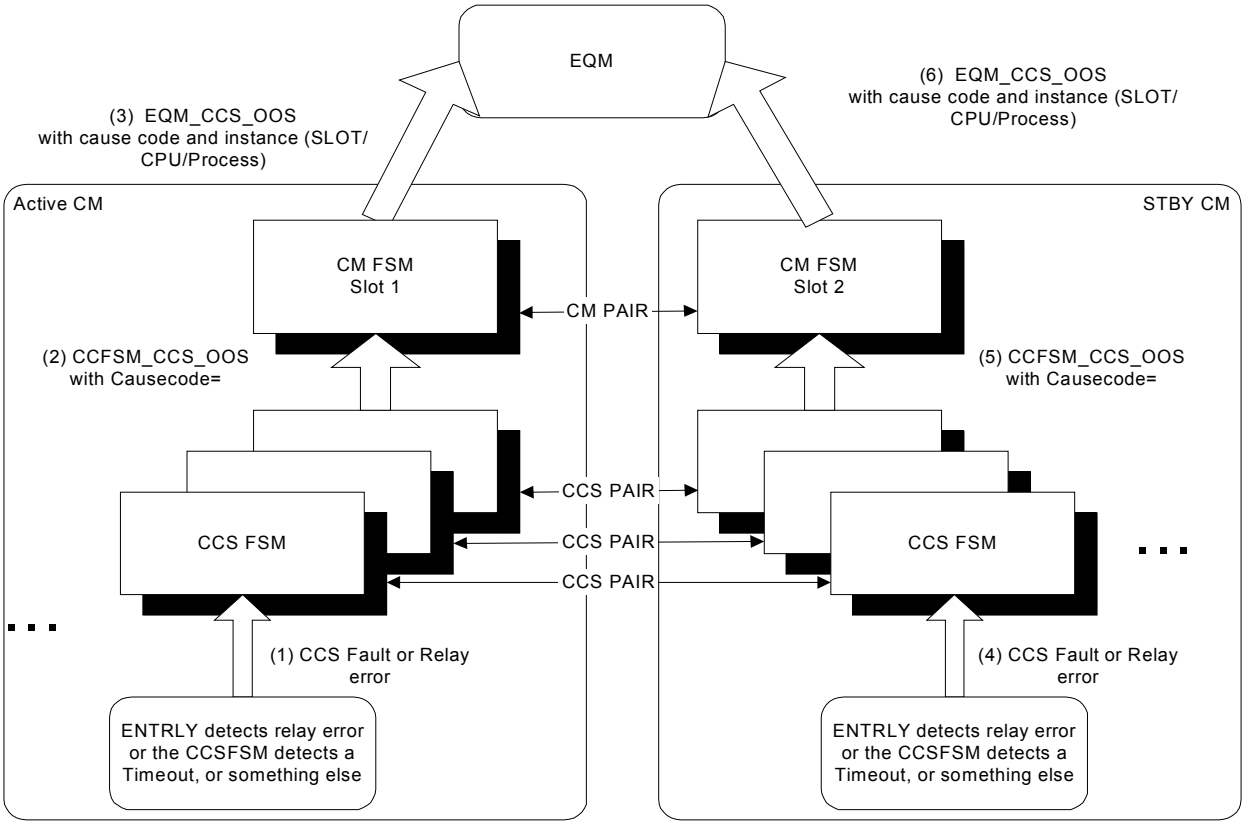


Figure 10-19

1) Active CCS FSM detects an active synced CCS going OOS – this could be by way of relay going down, or the CLAM on the CCS sending the SLAM a fatal alarm.
2) CCS FSM sends a CCSFSM_CCS_OOS with a cause code to the respective CM FSM.
3) CM sends CCS OOS event to the EQM. The EQM may initiate a fail over if required.

Items 4), 5) and 6) are the same except that this is from the standby CM. The EQM will decide if the whole CM is rebooted, or just certain processes.

10.7.2 EQM directed Failover

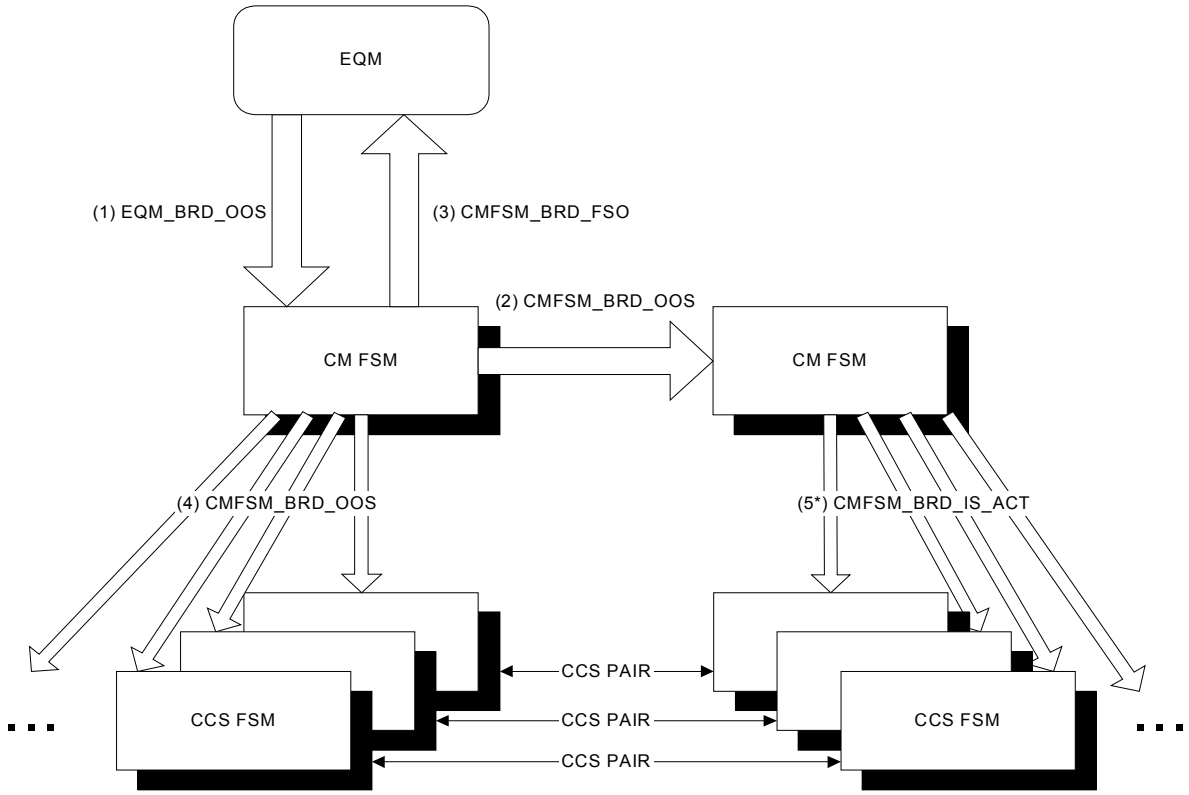


Figure 10-20

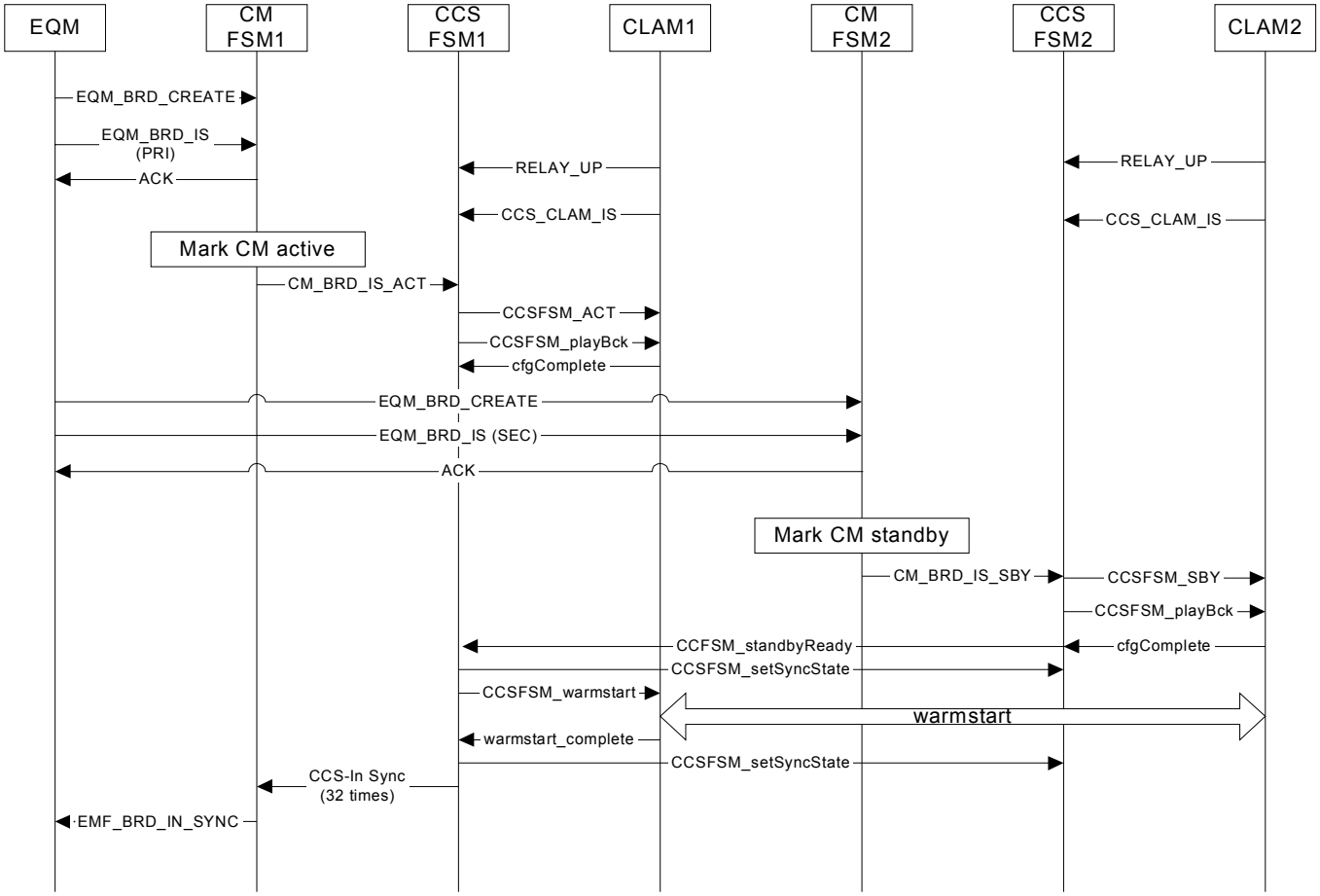
1) EQM detects CM board is OOS and sends the EQM_BOARD_OOS event to SLAM.
2) CM FSM sends the CMFSM_BRD_OOS event to the partner CM FSM
3) CM FSM informs EQM of the Failover
4) CM FSM sends the OOS event to all CCS FSMs for the defunct board.
5) Newly active CM FSM sends Active IS event to all the CCS FSMs for the new active CM. (*)Note that this happens asynchronously after (2), and so may happen much earlier.

10.7.3 Centralized Failover decision control

For both cases, the decision control to enable the failover is at the EQM. This centralizes the control .

- A future (post 5.1) TL1 command may be added to move a CCS from a CM pair to a new CM pair. Processing for this may change the Failover logic.

10.8 Sample startup message walkthrough



10.9 SLAM replication

SLAM replication component replicates events between active and standby processors.

States to be replicated:

- CM states,
- CCS states,
- TL1 events
- TL1 FSM state

The start of sync (before starting copying of DB) will block new config TL1 commands, wait for all outstanding TL1 FSMs to go to IDLE/terminate/timeout and then replicate all

the states. Runtime replication can go on in parallel. Will follow PSF strategy using sequence numbers.

Failover will include equipment provisioning/state audit.

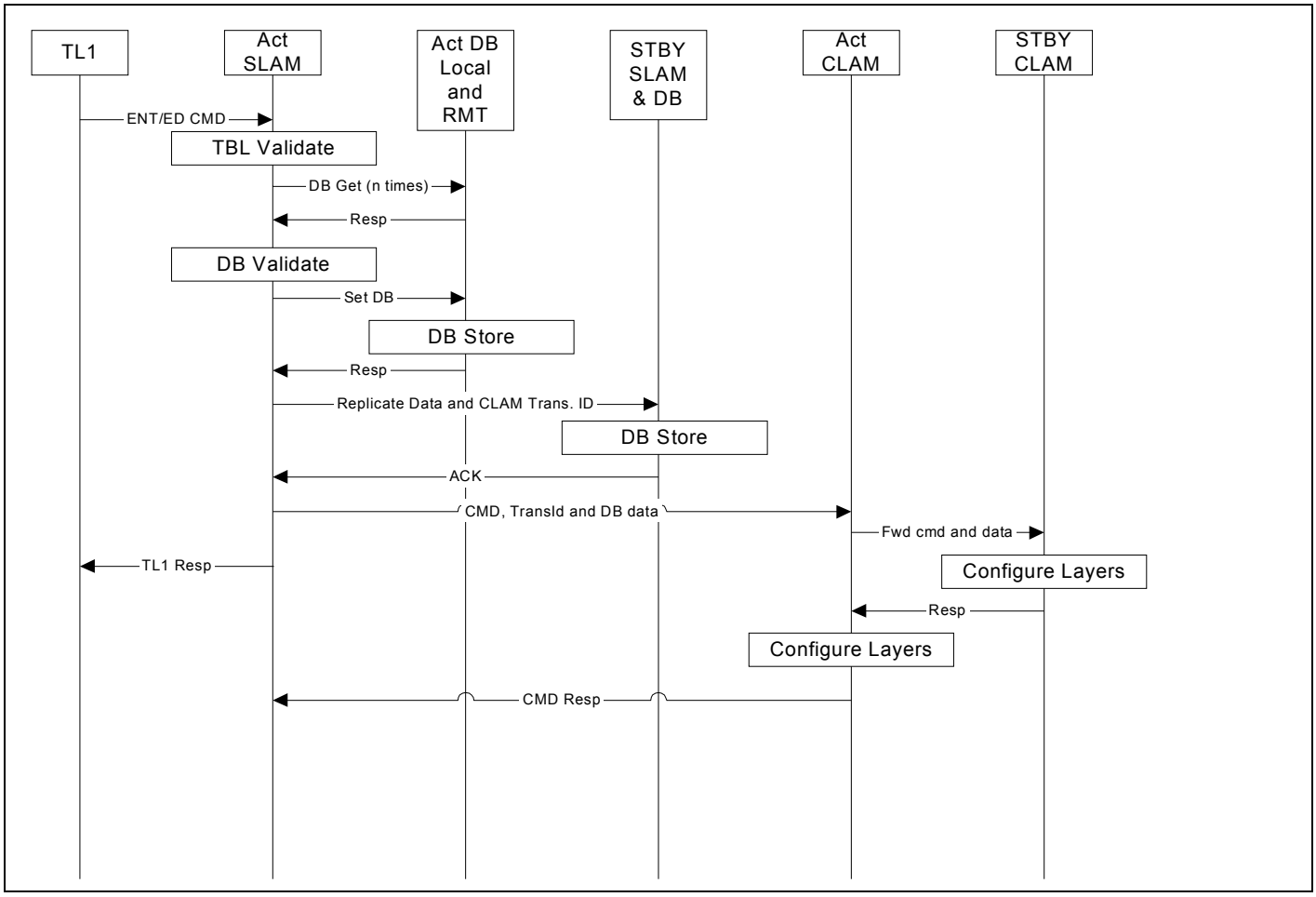
10.10 TL1 FSM

The basic support for TL1 provisioning and retrieve commands is shown in 8) TL1 Command Flows. This section will contain the detail flows.

10.10.1 ENT and ED Commands

This will depict the commands for both CCS and applications on the CCM. Data for the CCM is stored on the CCM. The SLAM on the SP will have a DB proxy that will automatically go to the correct disk for DB access. So the SLAM will be able to get and set data the same way independent of where the data is stored (locally or remotely on the CCM).

Note: There are 2 boxes for the Act SLAM(Act SLAM) and its Database (Act DB). This shows the details. For simplicity, the box marked "STBY SLAM & DB" contains both the SLAM and it's associated DB.

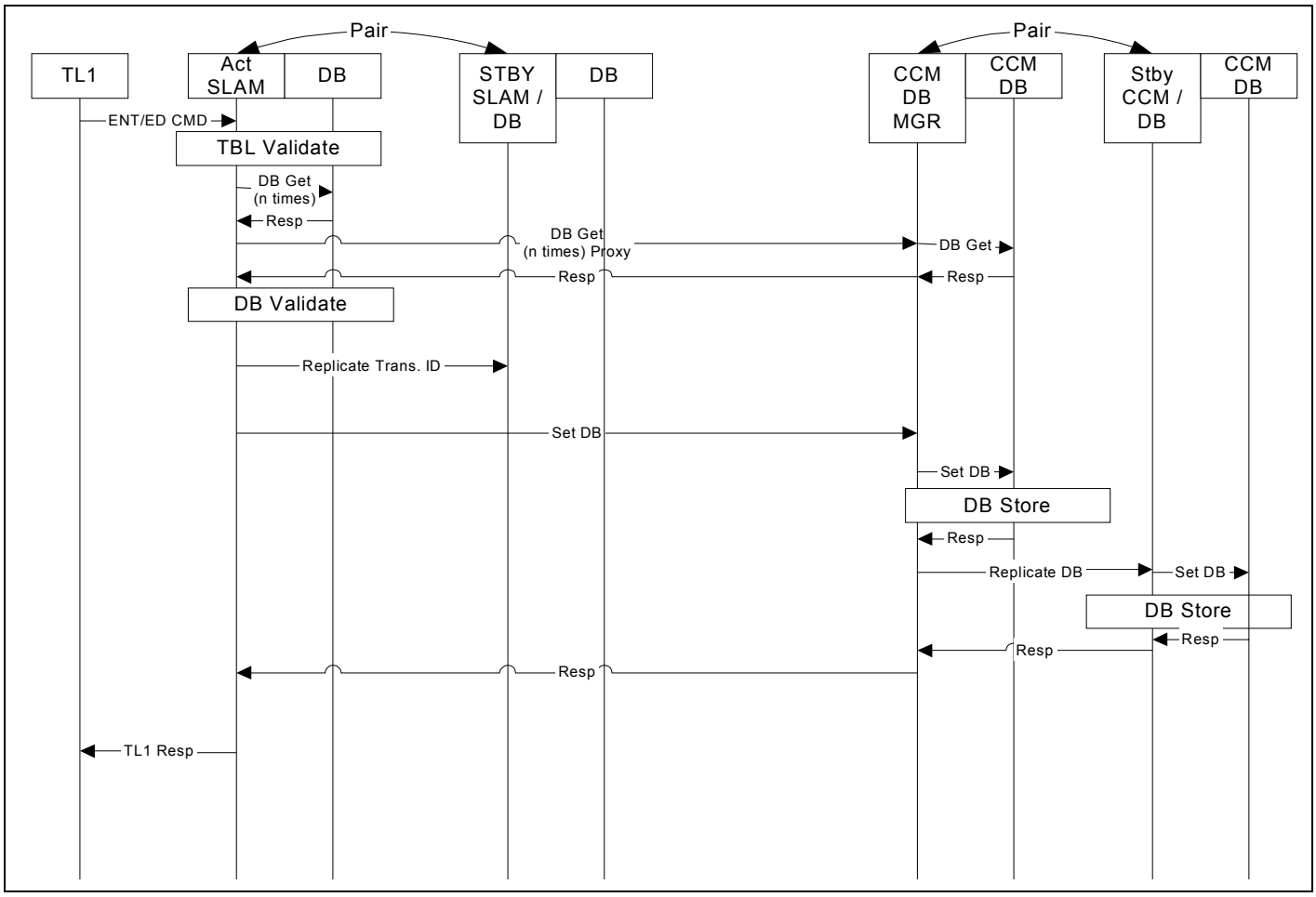


10.10.2 ENT/ED Commands Involving SP and CCM Based Data Stores

This flow shows the interaction when data is needed from the database on the SP and the CCM. This is similar to the regular ENT/ED flow except this shows the details with the CCM. The DB Manager only does the database IO and the replication to the standby CCM.

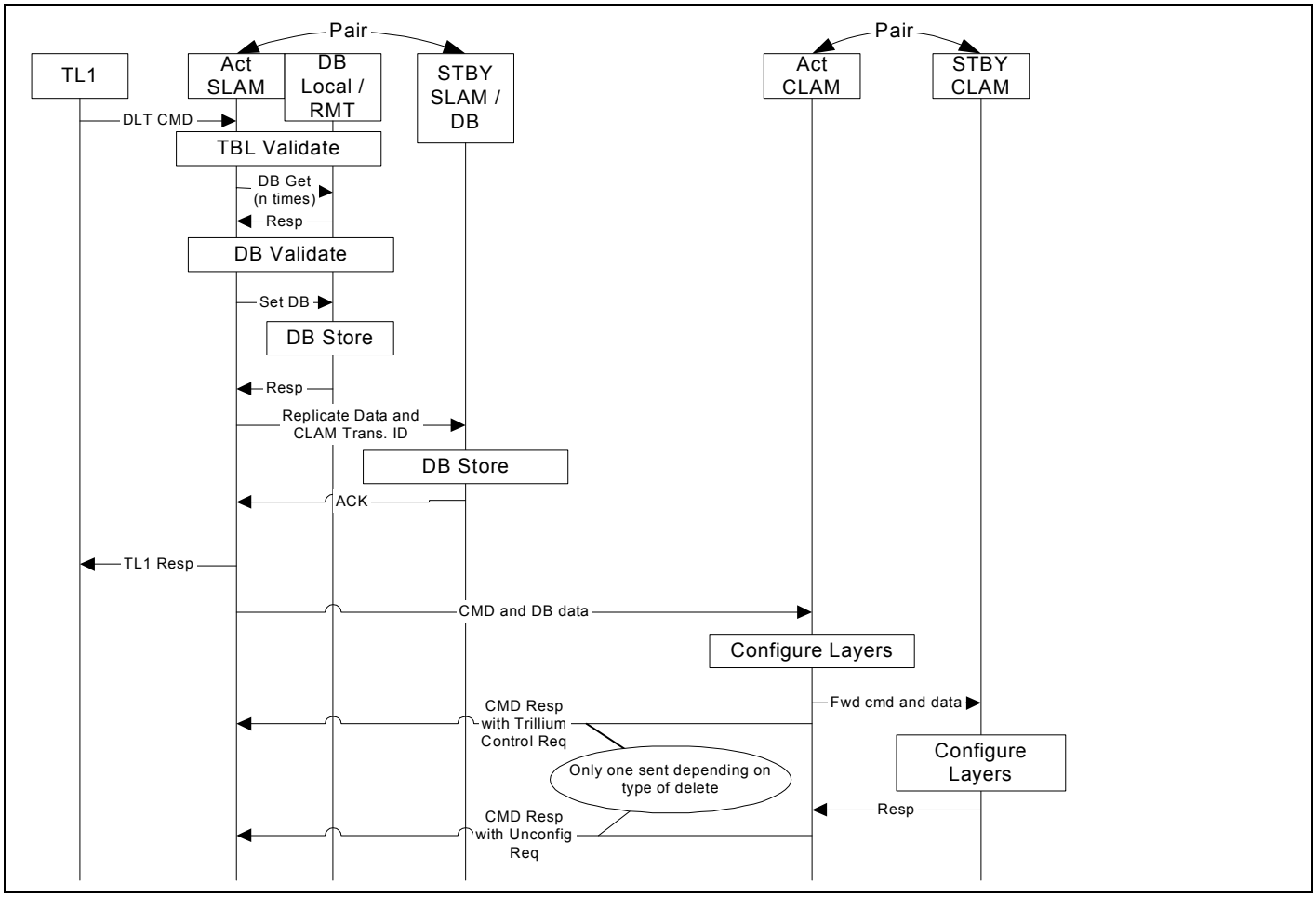
Note the replication module:

- The Active SLAM will replicate the data to the standby SLAM
- Active CCM DB manager will replicate the data to the standby CCM.



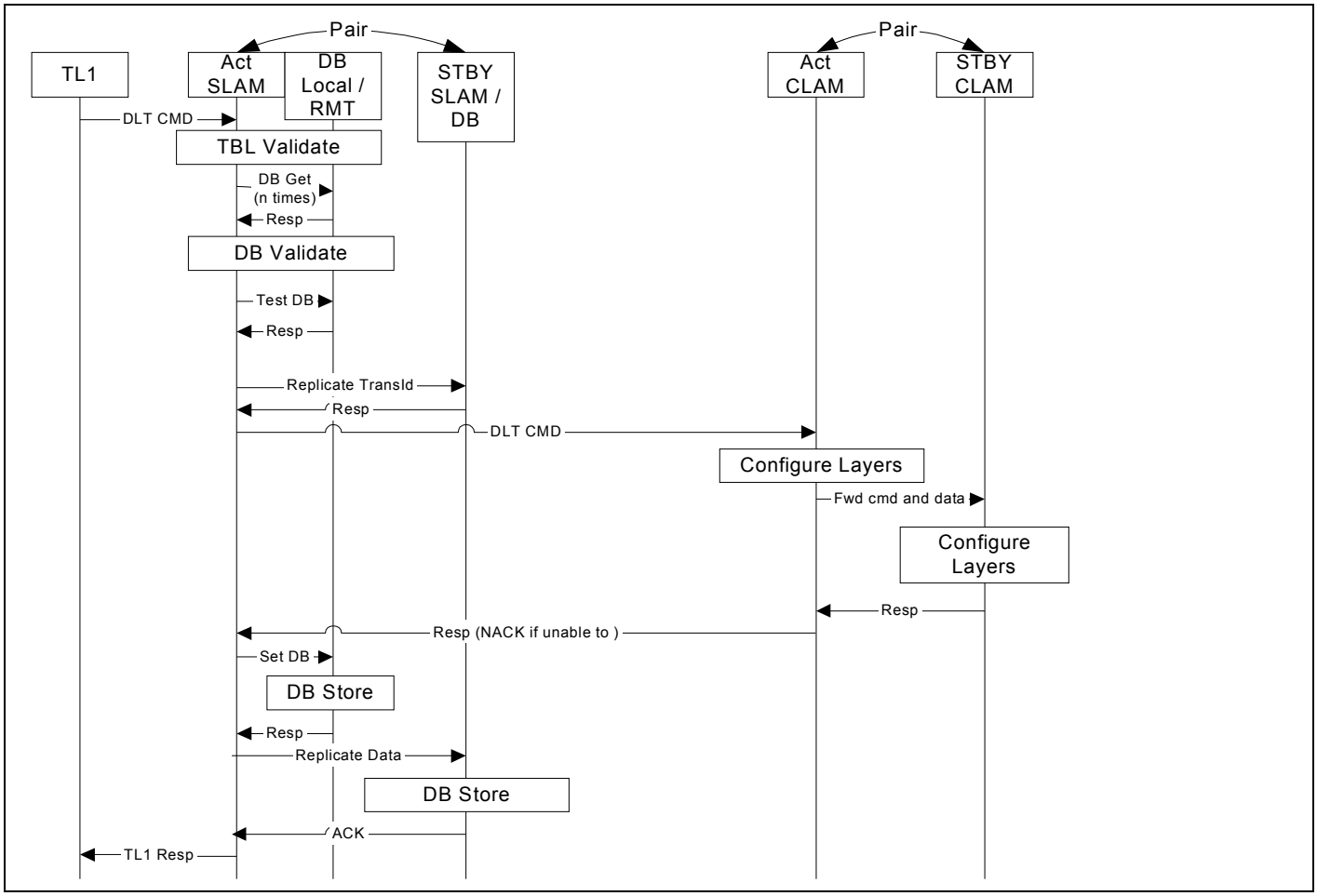
10.10.3 DLT Commands

The normal delete will just need DB validation. This diagram will show the local and remote data base together. This is possible since the remote DB access is handled under the covers with a loosely coupled call and are the same as 10.10.2.



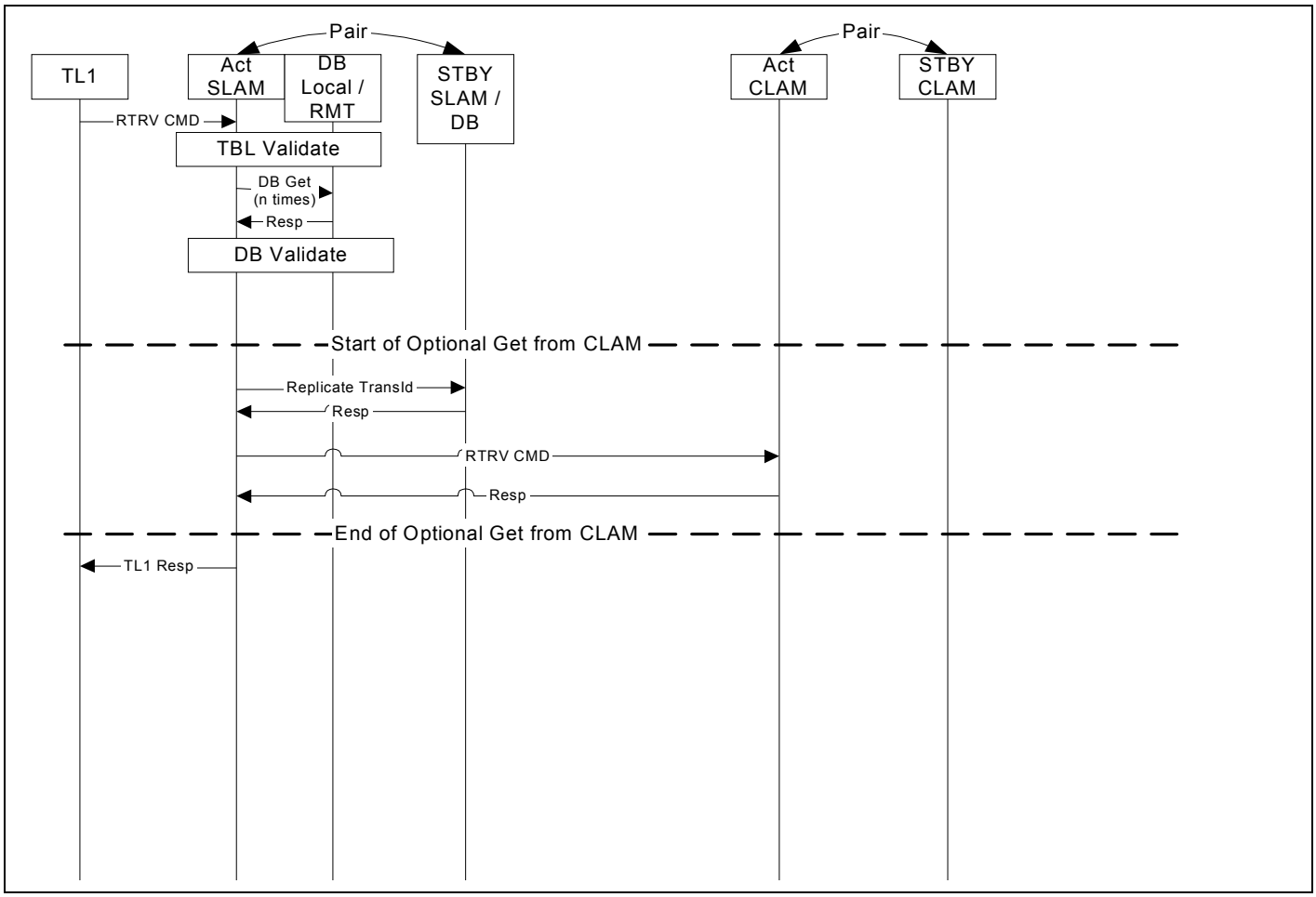
10.10.4 DLT Commands Requiring Validation from CLAM

These deletes need to be validated from the CCS or CCM. Cases like deleting CICs cannot happen if the calls are up on those CICs. This flow will show the request going down to the CLAM for validation.
Note: The Storage to the data base and the TL1 response is after the CLAM response. This is the only case of this delay.

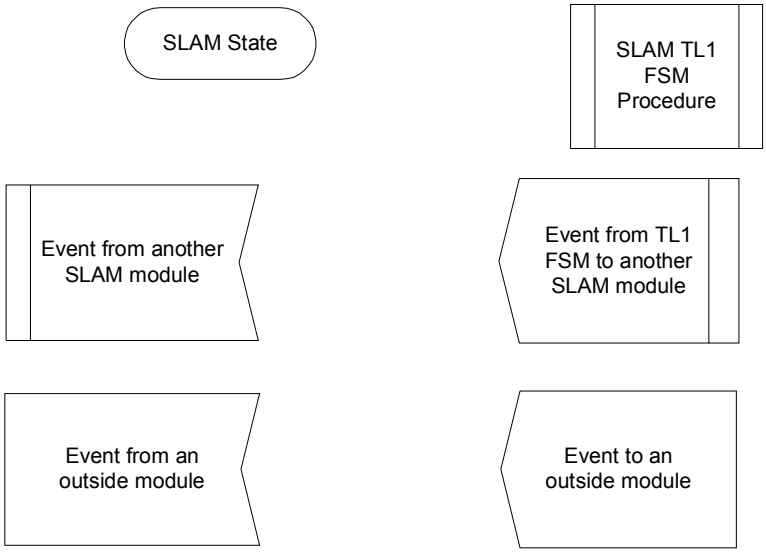


10.10.5 RTRV Commands

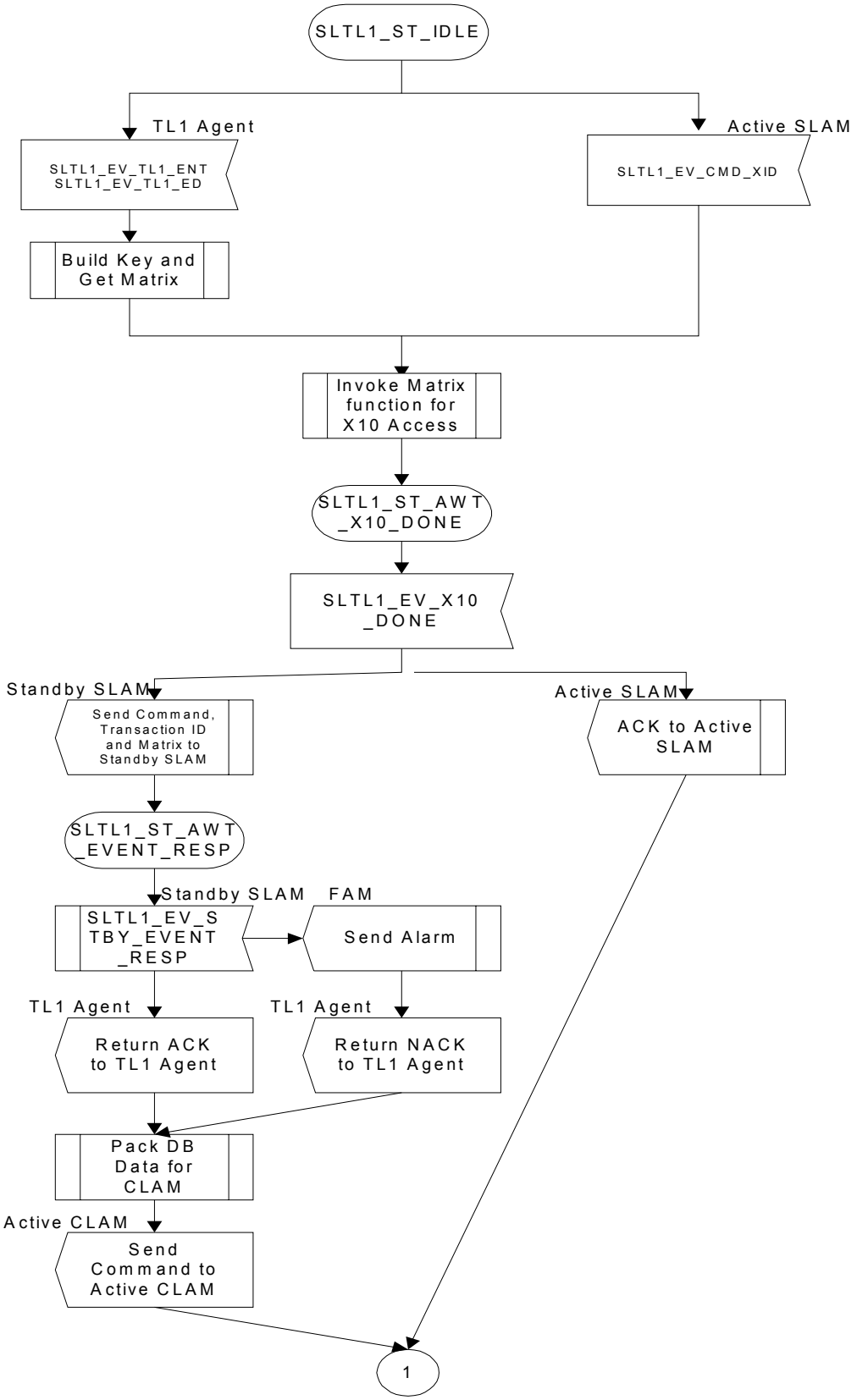
This diagram will show the RTRV flow. This may be just a get from a database. It also contains an optional request from a CLAM for run time state.

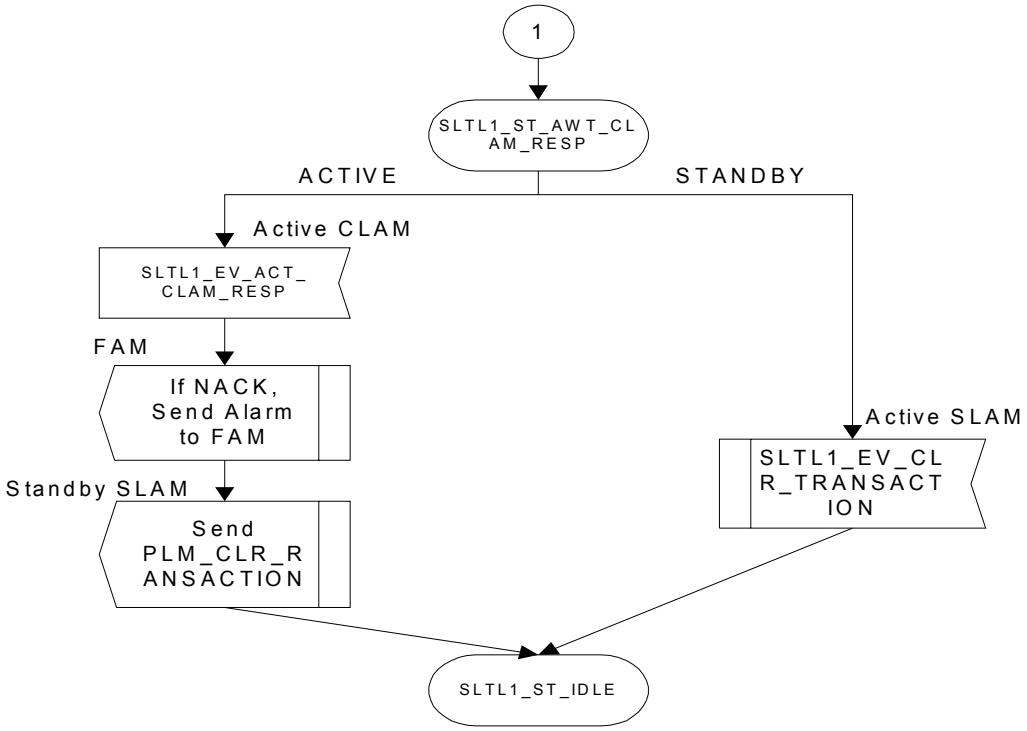


10.10.6 SDL Representation Guide

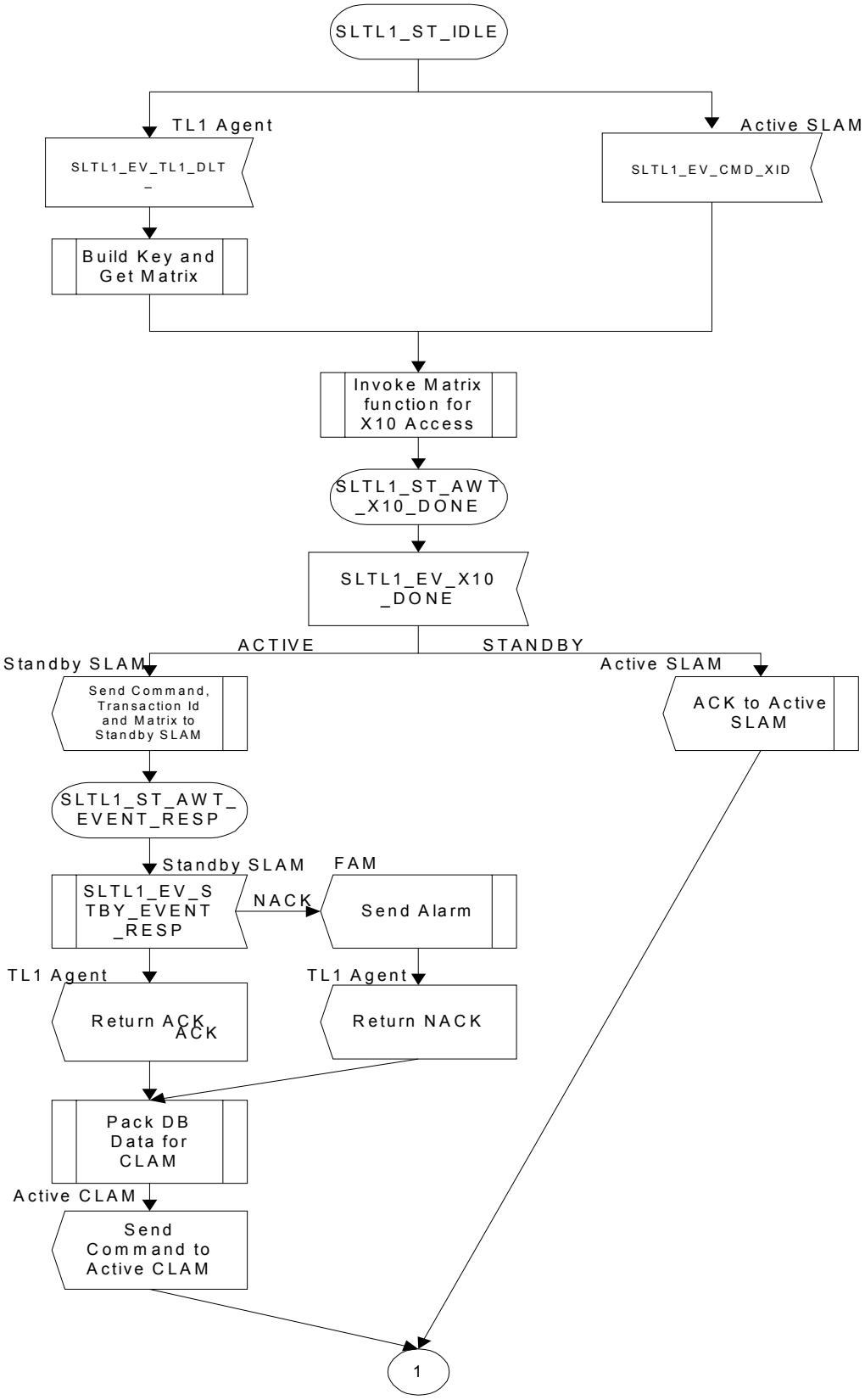


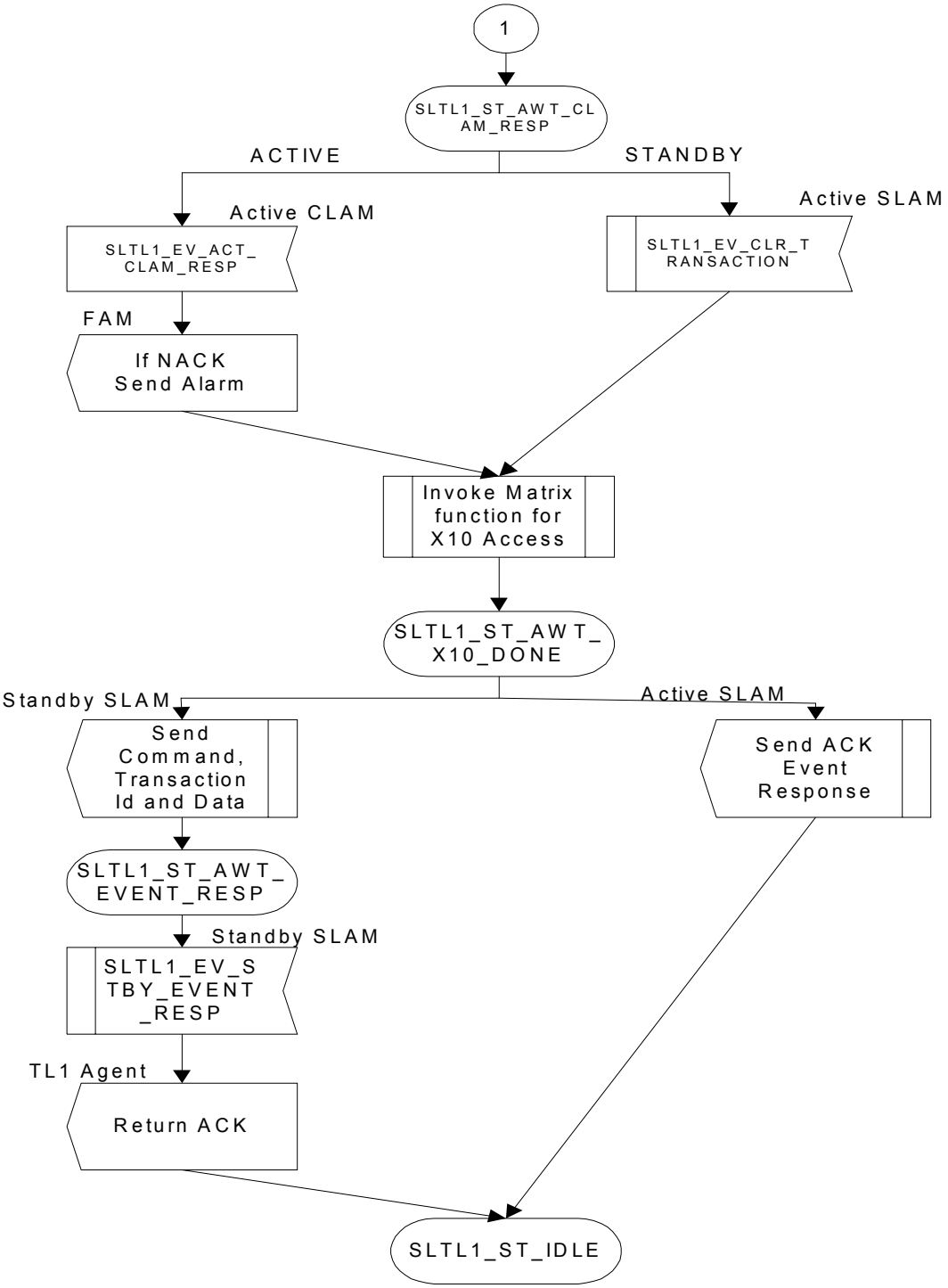
10.10.6.1 SLAM – ENT/ED Command



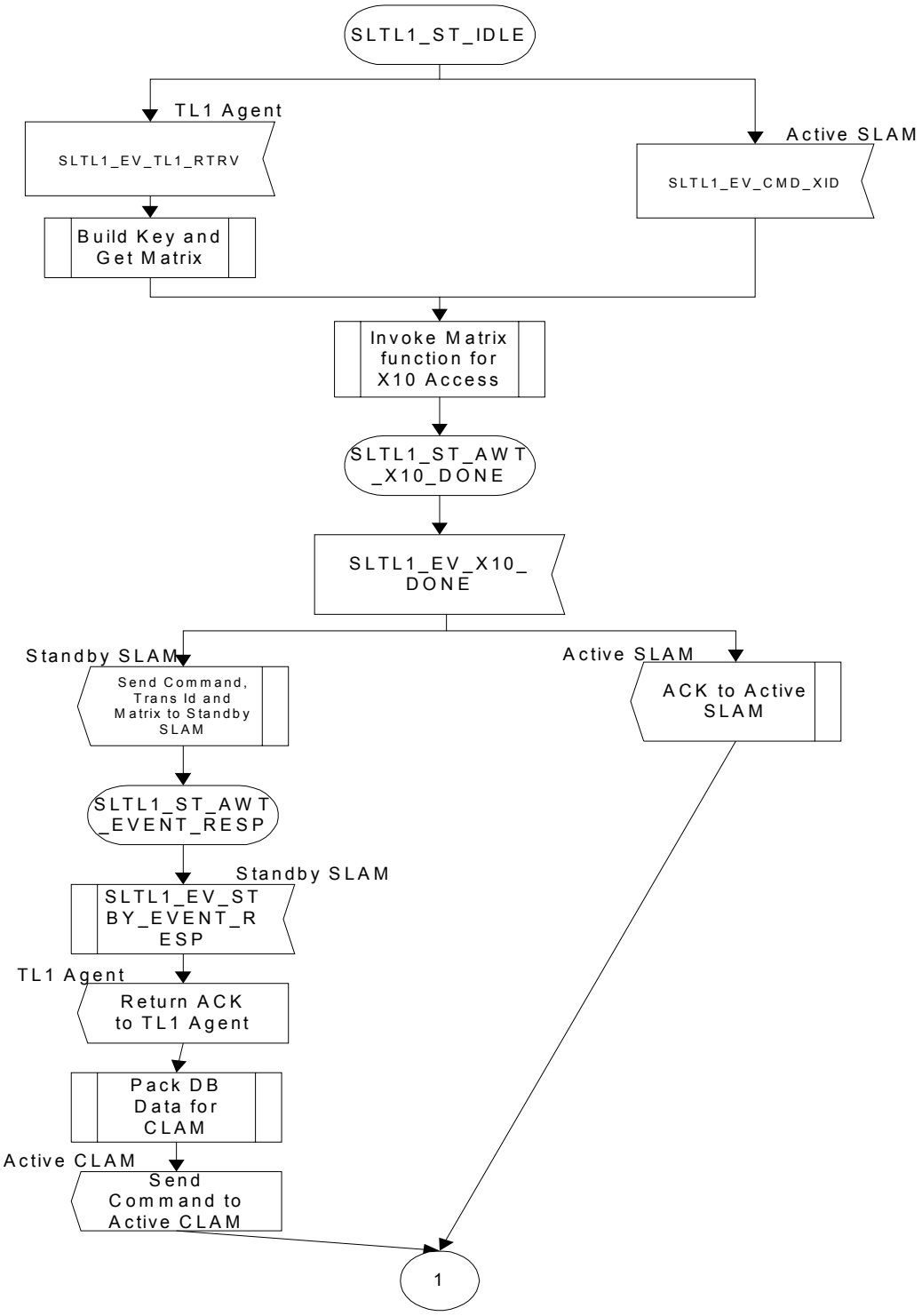

1
SLTL1_ST_AWT_CLAM_RESP
ACTIVE
STANDBY
Active CLAM
SLTL1_EV_ACT_CLAM_RESP
FAM
If NACK, Send Alarm to FAM
Standby SLAM
Send PLM_CLR_RANSACTION
Active SLAM
SLTL1_EV_CLR_TRANSACTION
SLTL1_ST_IDLE

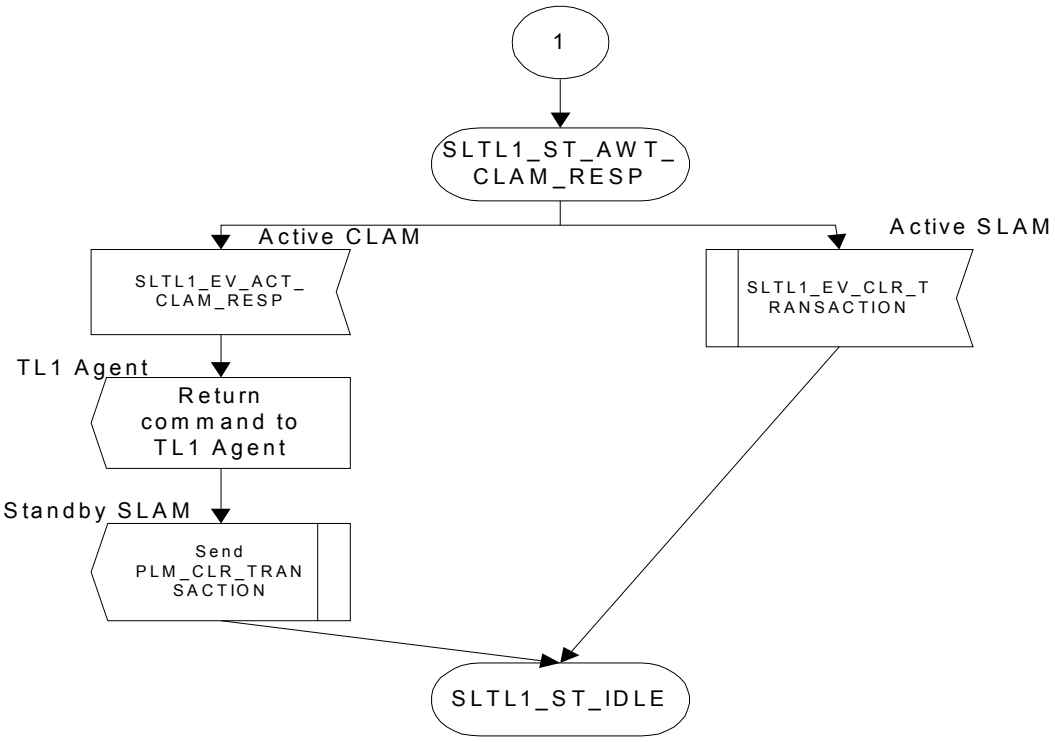
10.10.6.2 SLAM – DLT Commands Requiring Validation




1
SLTL1_ST_AWT_CLAM_RESP
ACTIVE
STANDBY
Active CLAM
Active SLAM
SLTL1_EV_ACT_CLAM_RESP
SLTL1_EV_CLR_TRANSACTION
FAM
If NACK Send Alarm
Invoke Matrix function for X10 Access
SLTL1_ST_AWT_X10_DONE
Standby SLAM
Active SLAM
Send Command, Transaction Id and Data
Send ACK Event Response
SLTL1_ST_AWT_EVENT_RESP
Standby SLAM
SLTL1_EV_STBY_EVENT_RESP
TL1 Agent
Return ACK
SLTL1_ST_IDLE

10.10.6.3 SLAM – RTRV Commands




1
SLTL1_ST_AWT_
CLAM_RESP
Active CLAM
Active SLAM
SLTL1_EV_ACT_
CLAM_RESP
SLTL1_EV_CLR_T
RANSACTION
TL1 Agent
Return
command to
TL1 Agent
Standby SLAM
Send
PLM_CLR_TRAN
SACTION
SLTL1_ST_IDLE

10.11 SLAM interface with EMF

10.11.1 EQM interface

This interface is similar to board events. The API is defined in components/telica_common/inc/eqp_evts.h. CM events are from EMF are:

- EQPEVT_BRD_CREATED: This is sent from EMF when a CM is created with a TL1 command or when booting. There is no confirm required.
- EQPEVT_BRD_DELETED: This is send when the user deletes the board. A verification (positive or negative) confirm is required (timed by EMF).
- EQPEVT_BRD_DELETE_QUERY: This requires a response back to EMF. If there is some provisioning dependent on this board, then a NACK is sent back.
- EQPEVT_BRD_STATE_CHANGE: A verification (positive or negative) confirm is required (timed by EMF).
- EQPEVT_BRD_SIG_STATE_CHANGE
- EQPEVT_BRD_FAILOVER: Sent in when the EMF has failed this board to the standby.
- EQPEVT_BRD_AUDIT_START: Called at startup and fail over. This is followed my multiple CREATEs and STATE_CHANGEs. The process should verify the state of the boards.
- EQPEVT_BRD_AUDIT_END: Terminates the AUDIT_START.
- EQPEVT_BRD_SIG_FAILOVER: this is sent back to EMF from a shared semaphore fail over notification.
- EQPEVT_BRD_SWITCHOVER
- EQPEVT_BRD_EDIT: Not needed for SLAM

The following are new events for 5.0 releases:

- EQPEVT_BRD_IN_SYNC: This is sent to EMF to indicate that the active board is in sync with a standby board.

The following are Alarms or Events to send to the FAM:

- EVT_CCS_OOS: This is sent to the EMF when a CCS fails. The EMF will determine if this was the active or standby and may initiate a fail over if it was an active and in sync. It could also decide to just restart the instance.
- EVT_RLY_FAULT: This is sent to the EMF when the SLAM detects a relay failure to either a CCS or the CCS has lost relay to the sub-processes.

Also SLAM process events: booting, active, standby etc

Question: SLAM does not have an API to receive CCS state changes for any state information that the cm_mon may have. Is this required? Currently SLAM relies solely on relay alarms. The CM Monitor spec does include a heartbeat with applications on the CM, and an event in case of lost heartbeat.

Answer: The SLAM will rely on RELAY alarms for status. If the Relay is up, then the CCS is alive. If it goes down, the CCS must be down and the SLAM will indicate to the EMF. The EMF will then tell the CM_MON (via a GoAhead message) either fault the card or restart the CCS.

10.11.2 FAM interface, Alarms

The detail alarm interface is defined in 1)79-5017, Media Gateway Controller (MGC) Architecture Specification under the section named "Communication fault policies".

10.12 SLAM interface with CLAM

The SLAM communicates with CLAM over the relay. The relay channel is via XU process. The API definition is in the style similar to the one defined between other protocol layers in …/signaling/lm/common/clslif.h.

The events defined between the SLAM and CLAM are as follows:

```
/* event types between CLAM and SLAM */
#define  EVTCLSLTL1          (1)  /* TL1 request from SLAM to CLAM */
#define  EVTCLSLPLAYBACKREQ  (2)  /* Playback req from SLAM to CLAM */
#define  EVTCLSLPLAYBACKCFM  (3)  /* Playback cfm from CLAM to SLAM */
#define  EVTCLSLSTSCHG       (4)  /* CCS State Change req SLAM to CLAM */
#define  EVTCLSLSTAIND       (5)  /* CCS Status Ind from CLAM to SLAM */
```

For the TL1 commands and Playback request the SLAM packs the individual table Ids into an mBuf and sends it to CLAM over the relay. On the CLAM side if the received request type indicates it's a TL1 command or a Playback request, the CLAM unpacks the number of tables indicated in message header and unpacks the tables one by one. The table type field indicates whether it's a dependent table from database required for processing the command or actual table modified through TL1.

For the status change APIs, the SLAM sends the old status and the new status to the CLAM. The status values are same as defined in the section 10.6.3. CLAM interprets these as GO_IS_ACT/IS_SBY requests.

10.13 Interface with Database manager

Separate thread.. Multiple contexts, loosely coupled DB APIs, cannot hold up SLAM while doing large retrieves or CCS bootups..

10.14 CCS ID Assignment and Table

There needs to be a common table to keep track of the what CCS goes with what MG. The outside world uses a mgName string and internal to the code, we use an ID. The SLAM will dole out the IDs on an algorithm defined in the MGC Arch. Spec 1)79-5017, Media Gateway Controller (MGC) Architecture Specification. The id of the CCS is generated from the following equation:

```
#define             NUMCCSPERSLOT 32
#define             NUMCCSPERCPU  8
#define             NUMIOSLOTS          17

ccsId = NUMCCSPERSLOT *(slot-1) + NUMCCSPERCPU *(cpu-1) + instanceId.
```

The instanceId is the number of the CCS on that CPU. Slot 9 could have a CM, but it never will since it has no backup. But we will use it to keep things generic. There will also be a pair of CCM in the system that this algorithm will overlap with, but this will be more generic.
So the first ccsId on slot 1, cpu 1 is 1 and the last ccsId on slot 17, cpu 4 is 544. ccsId = 0 is invalid. The CCS on the SP will be at the end of the table and allows for a CCS on either the master or slave.

The following table shows some of the assignments:

CcsId	cmSlot	cmCPU	InstanceId
1	1	1	1
2	1	1	2
3	1	1	3
4	1	1	4
5	1	1	5
6	1	1	6
7	1	1	7
8	1	1	8
9	1	**2**	1
80	3	2	8
513	17	1	1
544	17	4	8
545	sp-a	master	n/a
546	sp-a	slave	n/a
547	sp-b	master	n/a
548	sp-b	slave	n/a

Table 1

10.14.1 CCS Assignment Structures

A data structure will be kept by the SLAM to keep track of the CCS assignments and the associated mgName. There will need to be a hash table to quickly lookup the ccsId when a mgName is given. The data actually can be the slCcsCb pointer defined in 10.6.1 Data Structures.

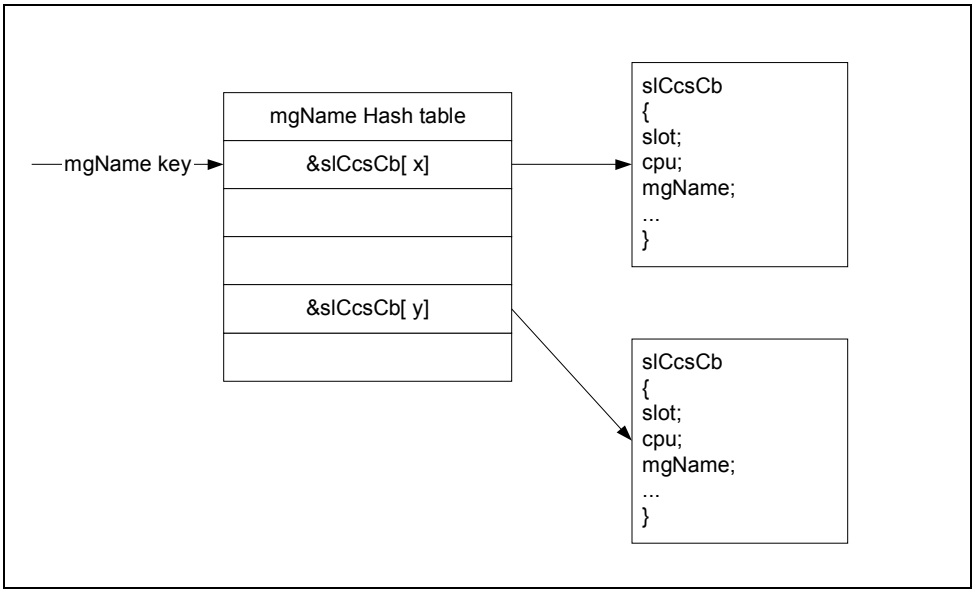


Figure 21

10.14.2 CCS Assignment Functions/Macros

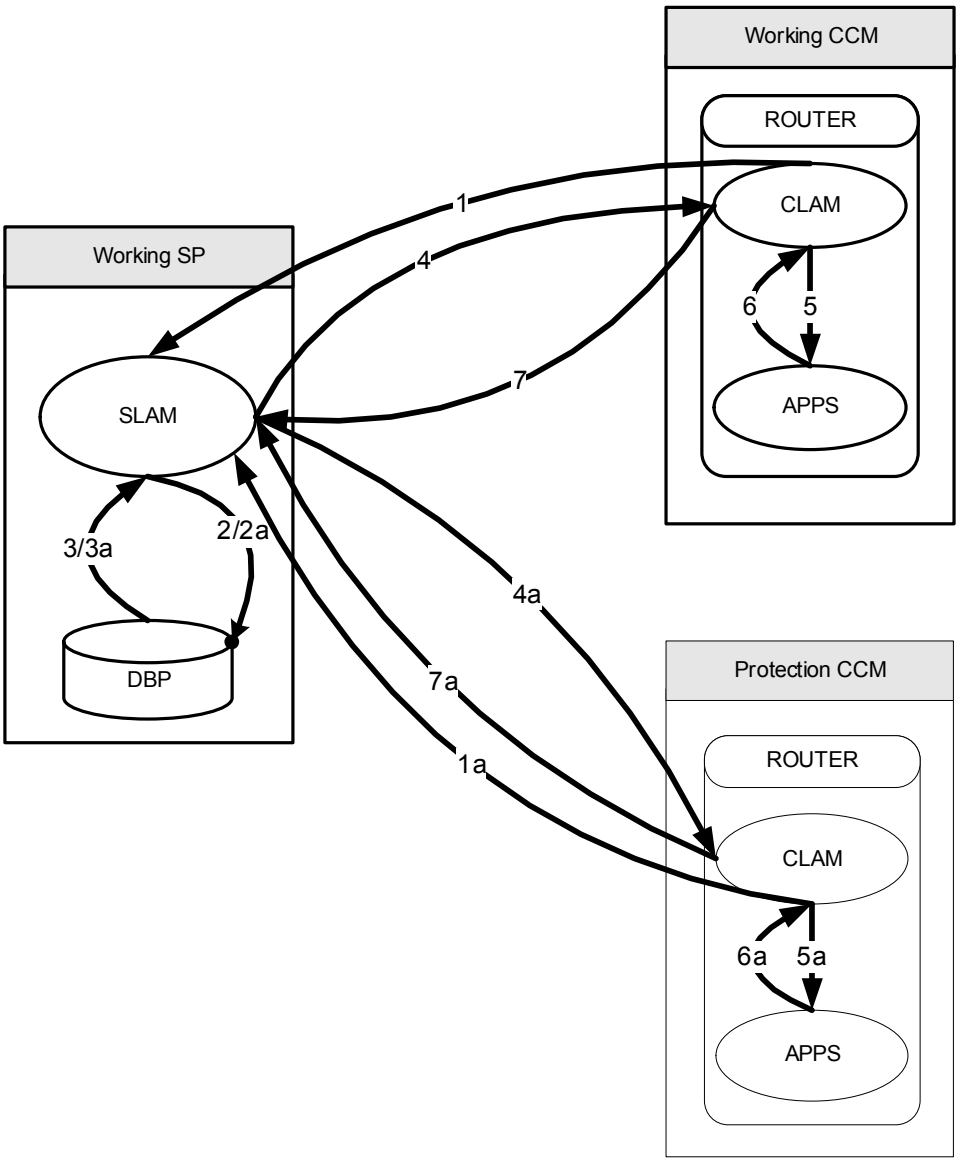
The following functions or macros will be required. They will be located in
Telica/components/signaling/lm/slam/base/sl_clam_util.c

- S16 slGetCcsIdFromMgName (char *name, U8 *ccsId)
  - This function will return the pointer to a ccsId that this mgName is assigned to
- S16 slClamAssignMg(MgcMgAssoc *mgcMgAssoc, U8 *slot, U8 *cpu, U8 *inst)
  - This is used when a new CCS must be assigned for the first time. The slot/cpu/inst are returned.
- S16 slGetCcsCb(U8 ccsId, SlCcsCb **ccsCb)
  - This will return the ccs Control Block given a ccsId.

10.15 LM rewind mechanism

This section describes the sequence of events for a CLAM rewind in a Distributed chassis. This is also the general or generic rewind mechanism for all clamTypes.

Distributed Chassis: SLAM-CLAM rewind



10.15.1 Rewind Event sequence

As detailed in the illustration above, here's the step-by-step break-up of events for SLAM-CLAM rewind.

10.15.1.1 SLAM events:

- **Step 1:** Active SLAM receives CLAM_IS event from Active CLAM
- ***Step 1a:*** *Active SLAM receives CLAM_IS event from Standby CLAM*
  - If XU, BRD are up for this CLAM, and mgAssoc flag is set to TRUE, sl_clam_bootup is called to trigger the rewind for this CLAM.
  - The clam state in clam_fsm is marked ACT_BOOTING/STBY_BOOTING correspondingly

- **Step 2:** Based on the Active clamType, SLAM gets the corresponding entry from SlClamBootDef matrix definition, and sends GET request to DB
- ***Step 2a:*** *Active SLAM sends GET request to DBP for standby CLAM tables*

- **Step 3:** DBP returns all rows for specified table and sends it back to SLAM.
- ***Step 3a:*** *DBP response for standby CLAM tables*
  - This data is collected cumulatively in context->boot.boottables

- **Step 4:** rewind data from context is copied to SlclPlbkReq in SlclMngmt and sent to Active CLAM in EVTSLCLPLBKREQ
- ***Step 4a:*** *Active SLAM sends EVTSLCLPLBKREQ to standby CLAM*

10.15.1.2 CLAM events

CLAM entry fn receives the EVTSLCLPLBKREQ:

- sends the CL_EV_BOOT_START event to cl_fsm
- cl_fsm does the pre_boot check to send state_change info, and triggers CL_EV_BOOT_READY event
- cl_fsm handles CL_EV_BOOT_READY event and kicks-off clam boot by calling clProcessBootCfgEvt
- clProcessBootCfgEvt in turn, sets lmEvt=LM_BOOT_EVT and the flag clCb.genCfgState=CLM_GEN_CFG_IN_PROCESS and kicks-off the clam boot mechanism.

Note: Rewind mechanism in CLAM involves executing the bootMt defined in ../common/lm_bdy1.c for the specific clamType.

LM_BOOT_EVT is handled by each module in the boot matrix ,

- corresponding table rows sent from slam in the plbkReq are unpacked,
- clam module fsm is kicked-off with cfg_ev.

- **Step 5/5a:** CLAM module FSM sends CfgReq to corresponding protocol Apps layer(s)

- **Step 6/6a:** Apps layer(s) send Cfm back to CLAM for the cfg

When each module is done processing the BOOT_EVT, it calls clModCallBack and control is sent to the next module in the boot matrix.

When all modules in the boot matrix are done processing the LM_BOOT_EVT:
- clModCallBack cleans up resources and calls clDeleteEvt,
- clDeleteEvt sets flag clCb.genCfgState = CLM_GEN_CFG_DONE and triggers the CL_EV_BOOT_END event to the cl_fsm
- cl_fsm receives the CL_EV_BOOT_END event and sets the clam state as CL_ST_ACT_STANDALONE/CL_ST_SBY_READY and calls cl_send_boot_resp

- **Step 7:** cl_send_boot_resp is responsible for sending EVTSLCLPLBKCFM from Active CLAM back to SLAM
- ***Step 7a:*** *Standby CLAM sends PLBKCFM to Active SLAM*

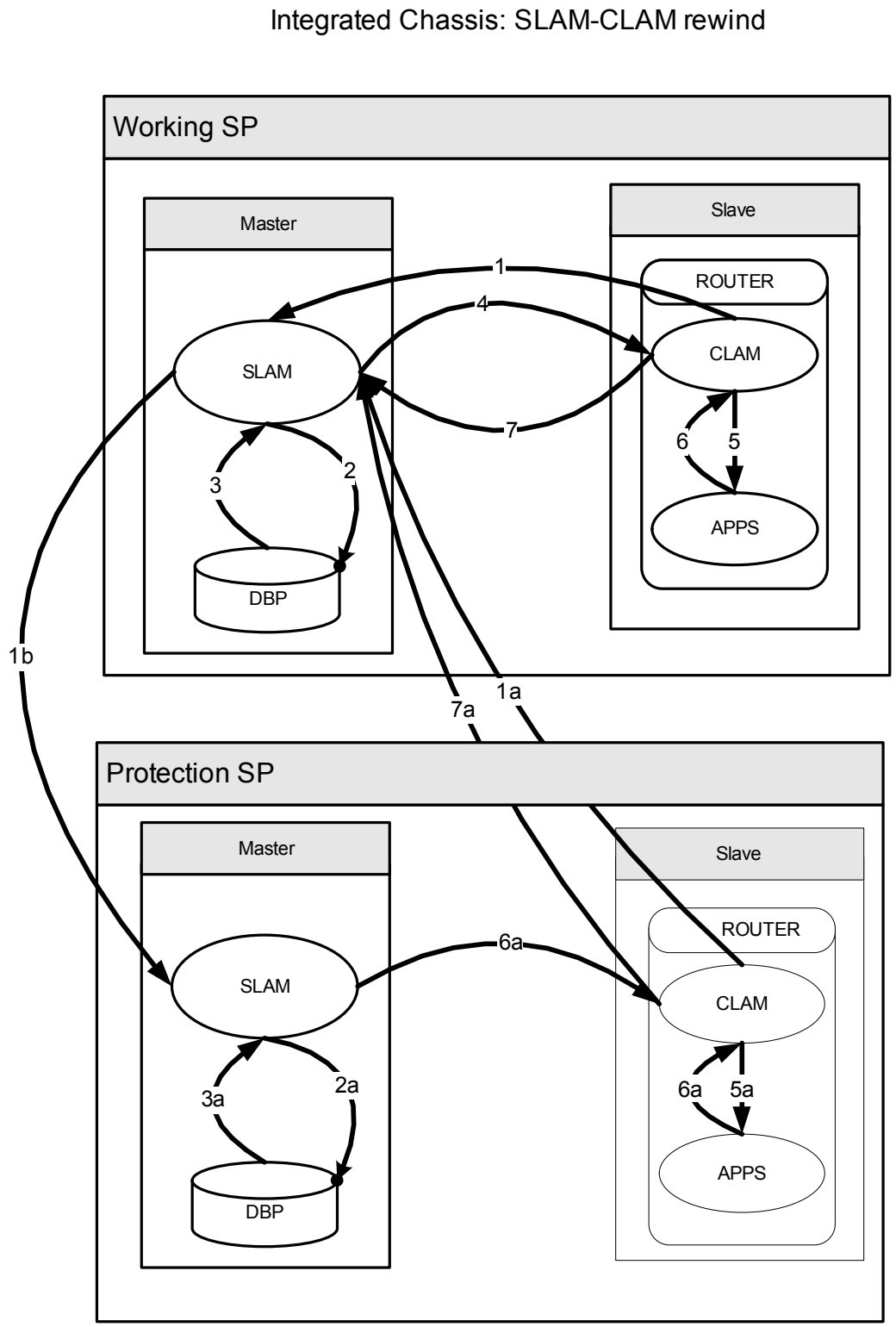
10.15.1.3 PLBKCFM handling in SLAM

Upon receiving EVTSLCLPLBKCFM,
- SLAM sends SLCLAM_EV_CLAM_PLYBCK_CFM event to the sl_clam_fsm
- sl_clam_fsm receives the SLCLAM_EV_CLAM_PLYBCK_CFM event and sends StChngReq to CLAMS, upon receiving the StChngCfg from CLAM, marks the corresponding clams as synced

This concludes the rewind machanism of CLAM.

10.15.2 LM rewind for Integrated Chassis

This section describes the sequence of events for a CLAM rewind in an Integrated chassis.



For SG_MGC_MG hot upgrades, CLAMs (ccs, router, etc ...) running on the standby SP slave will receive rewind data from SLAM on the Active. If there are changes in the tables, between the 2 versions, it will cause unpack errors, resulting in SP sync failure.

In order to overcome this issue, from release 5-1-2 onwards, standby SLAM is designed to send PLBKREQ to all standby CLAMs. However, sl_clam_fsm event processing is still done by the Active SLAM for both active as well as standby CLAMs.

This rewind sequence is similar to the list of events detailed in section 10.15.1. Only those special handling events specific to Standby SLAM, which are unique for SG_MGC_MG, are detailed below.

10.15.2.1 Special handling in Standby SLAM

- **Step 1b:** Standby SLAM (psf-slam) receives WARMSTART indication from Active SLAM

This event indicates that all Standby SP processes are ready, hence is a good time to trigger rewind for Standby CLAMs.

- **Step 2a:** Based on the standby clamType, standby SLAM gets the corresponding entry from SlClamBootDef matrix definition, and sends GET request to standby DB

- **Step 3a:** DBP returns all rows for specified table in the matrix and sends it back to standby SLAM.

- **Step 4a:** rewind data from Standby SLAM context is copied to SlclPlbkReq in SlclMngmt and sent to Standby CLAM in EVTSLCLPLBKREQ

10.15.3 Overview of Large table in DB

A DB table is loosely referred to as a 'Large Table', if there is a requirement to support 150K or more entries. For now, ROUTE-DIGITS is categorized as a 'Large table'.

This subsequent section describe the following rewind scenarios
- Large table rewind support in 6-2-1
- Design for Large table rewind support in 6-3

The 6-2-1 design for large table rewind enables supporting upto 300k route-digits. The Times10 datastore size is 192MB.

The 6-3 design will focus on optimizing this further, by using a new staged rewind mechanism. This will enable supporting more entries in the route-digits table.

The 6-3 requirement calls for supporting 1 million ROUTE-DIGITS. Times10 datastore limitations will restrict achieving this number, however, this design will make every effort to utilize the available resources and achieve the maximum possible route-digits entry support.

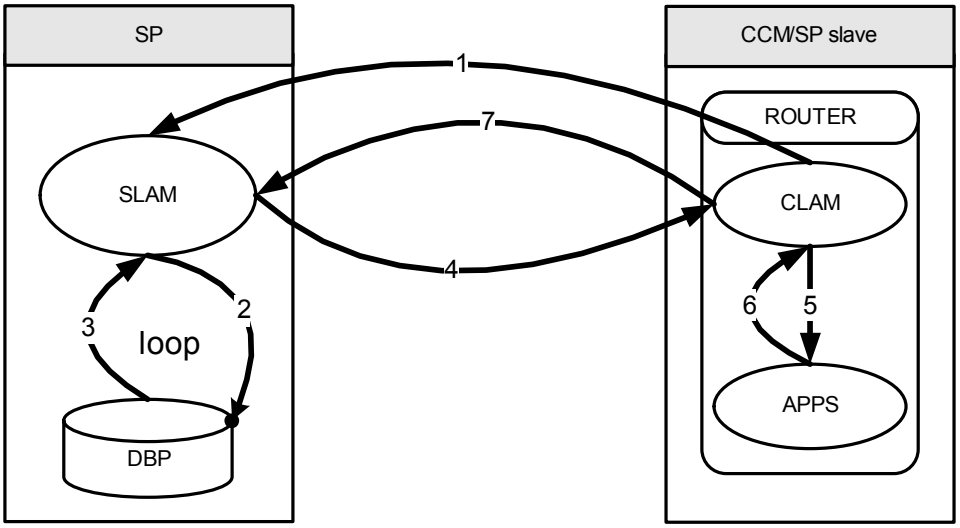
10.15.4 6-2-1: ROUTE-DIGITS large table rewind

This section describes the sequence of events for a large table rewind in release 6-2-1

In order to support ~250k entries, the ENT-route-digits is done via BULK-DOWNLOAD scripts. In case of boot-time rewind (in pre-6-2-1 releases), the SLAM/DBP seems to run out of memory, while trying to do a GET for ~250+ entries.

In order overcome this issue and to support 300k entries for ROUTE-DIGITS in release 6-2-1, a new rewind mechanism is introduced in SLAM.

6-2-1: ROUTE-DIGITS Large table rewind



This rewind sequence is similar to the list of events detailed in section 10.15.1. Only those special handling events specific to SLAM and DBP, which are unique for large table rewind scenarios are detailed below.

10.15.4.1 Special handling in SLAM

- **Step 2:** Based on the Active clamType, SLAM gets the corresponding entry from SlClamBootDef matrix definition, and sends GET request to DB

For those tables that are identified as 'Large tables': (Eg: ROUTE-DIGITS)

- bootKeyModFn is defined in the SlClamBootDef matrix entry
- This bootKeyModFn is automatically executed while processing that matrix entry
- A specialAction flag is set in the DB request to fetch the rowCount and unique MINROWID and MAXROWID, for large table. (***Ref: section 10.15.3.1.1)***
- Sl_clam_boot fsm is modified to recursively send GET requests for this large table, until all rows are RTRVd
- Subsequent GET requests to DB for this table are sent with specific range for RTRV, based on the ROWID ranges.
- SLAM will accumulate all the rows returned from the multiple iterations, perform version mapping if required.

When all rows are received from DB, the entire data is copied to SlclPlbkReq in SlclMngmt and sent to Active CLAM in EVTSLCLPLBKREQ

10.15.4.1.1 ROWID in DB

TimesTen database assigns a unique ID, called ROWID to each row stored in a table.

- ROWID is of type binary(16).
- The ROWID value can be retrieved through a pseudo column named ROWID.
- ROWID is not a real column, it does not require database space and cannot be updated,
  indexed or dropped.
- ROWID value persists throughout the life of the table row, but the system can reassign
  the ID to a different row after the original row is deleted.
- Zero is not a valid value for ROWID

Since ROWID is not a real column, it is not displayes via SQL queries like SELECT. However, it can be queried directly via ttlogin

Sample ROWID from Times10:

```
Command> select min(rowid) from rtdigits;
< 001BB664000000000000000000000000 >
1 row found.
Command> select max(rowid) from rtdigits;
< 001BBA0C000000000000000000000000 >
```
1 row found

This min(rowid) and max(rowid) represent the min and max range for the specified RtDigits table. These values are used in the iterative RTRV mechanism of large table rewind, as detailed in section 10.15.3.1 and section 10.15.3.2.

10.15.4.2 Special handling in DBP

In order to support the large table rewind mechanism, the following DBP changes have been implemented:

- New table BULK_PLBK_COUNT_TAB_ID is defined, for large table rewind specific date used for the multiple iterations
- SpecialAction handling in DB Proxy to return rowCount, MINROWID and MAXROWID, for first request from SLAM. (***Ref: section 10.15.3.1.1)***
- SpecialAction handling in DB Proxy to return matching rows from 'large table' based on the range specified by SLAM, in the iterative GET requests
- New DB Apis for each of the above mentioned special actions.

- **Step 3:** DBP returns matching rows from the specified range, back to SLAM.

Note: This recursive GET request sequence is shown in the illustration as a loop of steps 2 and 3.

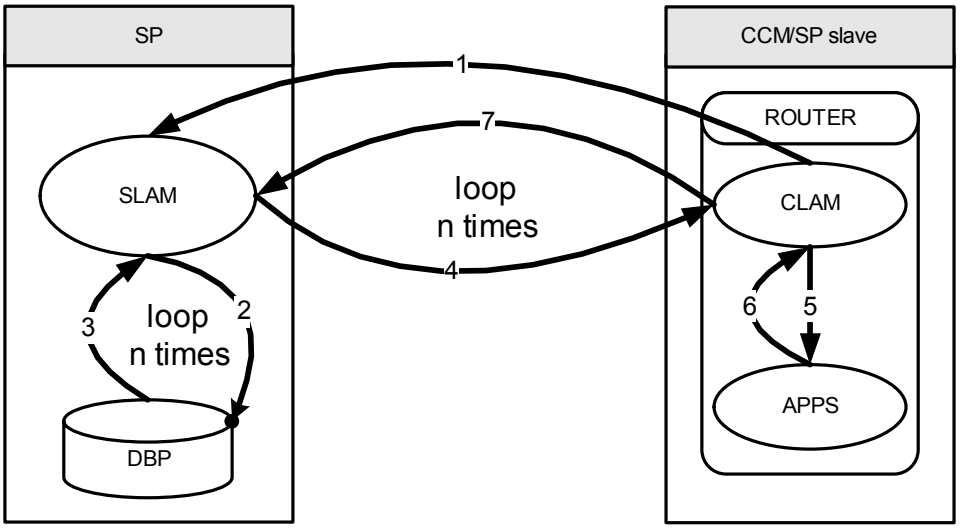
10.15.5 6-3: ROUTE-DIGITS large table rewind

This section describes the design to handle large table rewind for release 6-3.

For Release 6-3, the Feature Plan doc in QDI:31368 has a list of all requirements supported. This design doc will address only the following requirement

7682 ENT-ROUTE-DIGITS and DB table: RTDIGITS the entries shall be limited to: 1,000,000
6-2-1 limit: 150,000

6-3: ROUTE-DIGITS Large table rewind



This rewind sequence is similar to the list of events detailed in section 10.15.1. Only those special handling events specific to SLAM, CLAM and DBP, which are unique for large table rewind scenarios, in Release 6-3, are detailed below.

10.15.5.1 Infrastructure Change Summary

In 6-3, the 'Large table' rewind mechanism is further extended beyond the initial support implemented in 6-2-1.

The key features in 6-3 are

- Staged rewind, or multiple PLBKREQ events from SLAM, will be sent to ROUTER clam.

Both SLAM and CLAM infrastructure changes are required to support this functionality.

10.15.5.2 Staged Rewind Mechanism

In order to achieve a staged rewind mechanism, the SlClamBootDef matrices for ROUTER clam need to be broken down without compromising on functionality.

This is achieved by essentially, breaking this matrix into following stages

- **Stage 1:** This stage will comprise of tables, in the current ROUTER clam matrix, that occur before ROUTE-DIGITS. All these tables will be sent as a single PLBBKREQ

- **Stage 2:** This stage comprises of the ROUTE-DIGITS . Each iterative GET response from DB will trigger a PLBKREQ, until all rows are rewound.

  Note: Each iterative ROUTE-DIGITS rows from DB calls for a PLBKREQ in the staged rewind, hence we loop in this stage until all rows are exhausted.

As the ROUTER-DIGITS rewind in CLAM does not need other dependent tables, a standalone stage for this table will not affect functionality.

- **Stage 3:** This stage comprises of all tables, in the current ROUTER clam matrix, after ROUTE-DIGITS

-→ **stage 1:** following tables will be in stage 1

LM_RSTGN
LM_FTH
LM_DGTMOD_DN
LM_GENIE
LM_DGTMOD_GEN
LM_DGTMOD_PARAM
LM_DGTMOD_TNS
LM_TREATMENT
LM_DGTSCREEN
LM_RTLST

----→ **stage 2:**
LM_RTDIGITS

-→ **stage 3:**

Remaining tables.

Note: This recursive PLBKREQ/PLBKCFM sequence is shown in the illustration as 'loop n times' in steps 4 and 7.

10.15.5.2.1 Future Impact of Staged Rewind

In general, the staged rewind mechanism defines specific stages for rewind, such that, each stage is self-sufficient with the data/tables required for configuring the underlying protocol layers. This rule applies to all clamTypes.

- **Support for new tables**

The break-up of stages is done based on table dependencies, in other words, by taking into account the dependent configuration data needed prior to the rewind of a particular table. Based on this information, the tables are ordered and eventually broken-up into stages.

Due to the inherent dependencies in ordering, when new tables are added in the future, it is necessary to preserve the integrity of this data and the rewind stages.

- **Support for Pipeline mechanism in LM rewind**

The introduction of stages in the LM rewind mechanism will allow extending this further, to support pipeline mechanism in handling rewind events.

In other words, there is a proposal to enhance the sl_clam_boot fsm to handle events in a pipeline fashion, processing events as they are received. This will enable

- processing multiple plbkreq/plbkcfm in parallel
- fsm need not wait for one event sequence to complete before handling another. It can handle DB cfm events and Cfm from CLAMs in parallel.

The introduction of stage matrices and stage counters will enable achieving this pipeline handling.

Note: However, this will be a future possibility and will not be implemented in 6-3.

10.15.5.3 Special handling in SLAM-CLAM Interface

As the staged rewind mechanism triggers multiple PLBKREQ/PLBKCFM, state transitions triggered by these events, in both SLAM and CLAM will have to be put on hold until the final REQ/CFM is received.

In order to achieve this, and to keep track of the different stages in the rewind mechanism, stageCounter and stageNum are introduced in the PlbkReq interface structure.

../lm/common/slclif.h

```
/* Playback request structure */
typedef struct _SlclPlbkReq
{
```

```
   U16 numTables;
   CmTbCb tblLst;
   U8  stageCounter;  → new in 6-3
   U8  stageNum;      → new in 6-3
} SlclPlbkReq;

/* Playback response structure */
typedef struct _SlclPlbkResp
{
   S32 tableId;
   S32 errorCode;
   U8  stageNum;      → new in 6-3
} SlclPlbkResp;
```

Note: The corresponding pack/unpack functions that process SlclMngmt will be updated to account for the new parameters.

10.15.5.3.1 ROUTER clam boot matrix changes

Router CLAM boot matrix definition in ../lm/common/lm_bdy1.c will be split as well, to match the staged SlClamBootDef matrices.

CLAM will use this matrix to process LM_BOOT_EVT, so the PLBKREQ tables from SLAM should match with the boot matrix in order to successfully complete the rewind mechanism.
So this will be split in 3 as well, similar to the stages detailed in section 10.15.4.2

10.15.5.4 Special handling in SLAM

The following are special handling changes in SLAM, for sending EVTSLCLPLBKREQ to CLAM

- SLAM will set the stageCounter in the plbkReq based on the number of stages for the specific clamType
- Also, for each PLBKREQ sent, SLAM will incerement the stageNum in the plbkReq struct. This will help keep track of the number of stages.
- When SLAM has completed processing the final stage of matrices for ROUTER CLAM, the PLBKREQ being sent to CLAM will be the final request, in this case stageCounter=stageNum.

Upon receiving EVTSLCLPLBKCFM from CLAM, compares stageNum from SlclPlbkResp

If stageNum != stageCounter

- SLAM will skip sending SLCLAM_EV_CLAM_PLYBCK_CFM to sl_clam_fsm. ROUTER clam state in sl_clam_fsm will remain ACT_BOOTING/STBY_BOOTING until the final PLBKCFM is received

Note: As the CLAM boot timeout in sl_clam_fsm is now set to 30min, there will be sufficient time to process the multiple PLBK events.

Else

- SLAM is processing the final PLBKCFM from ROUTER CLAM
- SLAM sends SLCLAM_EV_CLAM_PLYBCK_CFM event to the sl_clam_fsm
- sl_clam_fsm receives the SLCLAM_EV_CLAM_PLYBCK_CFM event and sends StChngReq to CLAMS, upon receiving the StChngCfg from CLAM, marks the corresponding clams as synced

This concludes the rewind machanism of CLAM.

Note: The staged rewind mechanism, although specific to the ROUTER clam in 6-3, is generic to all clamTypes. For non-ROUTER clamTypes, there will be a single stage during rewind, all relevant table data will be handled in this single PLBKREQ.

10.15.5.5 Special handling in DBP

The DBP special handling to support Large table rewind involves handling recursive GET requests from SLAM. This is similar to the mechanism detailed in section 10.15.3.2 which is supported in 6-2-1.

Note: This recursive GET request sequence is shown in the illustration as ‘loop n times’ in steps 2 and 3.

This same infrastructure will be used for 6-3 as well.

10.15.5.6 Special handling in CLAM

The following are special handling changes in CLAM

CLAM entry fn receives the PLBKREQ

If stageCounter != stageNum

- cl_fsm will be modified to process the multiple PLBKREQ. In other words, boot_start event will be handled in act_booting as well as standby_booting as well.

- Steps 5 and 6 in the illustration is similar to regular rewind. After all tables are rewound in the current stage, clModCallBack will call clDeleteEvt

- clDeleteEvt will skip setting genCfgDone flag, also skip sending BOOT_END event to cl_fsm. Instead, it will increment the stageNum parameter in SlclPlbkResp struct and call cl_send_boot_resp to send PLBMCFM to SLAM.

Else

- CLAM is processing the FINAL PLBKREQ

- clDeleteEvt sets flag clCb.genCfgState = CLM_GEN_CFG_DONE and triggers the CL_EV_BOOT_END event to the cl_fsm

cl_fsm receives the CL_EV_BOOT_END event and sets the clam state as CL_ST_ACT_STANDALONE/CL_ST_SBY_READY and calls cl_send_boot_resp

10.16 IP Forwarder (IPF)

10.16.1 Summary

The IP Forwarder is a kernel driver that is run on the SP slave that will forward IP packets to and from the individual CCSs. The IPF will use the IP destination port to decide where to send the packet. Each CCS should have it's own port per protocol. The SLAM will need to program the forwarder with the destination port, active CM slot, and CPU. The forwarder (along with the IP stack) will take this info and forward the packet to the correct vpi/vci on the switch fabric. The IP forward API is defined in Reference 5) 79-3322, MGC IP Forwarding Functional Specification, D. Dafoe.

The SLAM cannot talk to the IPF directly since the SLAM is on the master SP CPU and the IPF is on the slave SP CPU and is a driver. The IPF is programmed via ioctls. Therefore the SLAM will send a relay message to a slave process (CCS) that will in turn generate the ioctl. The IOCTL return will just be packaged up and sent back to the SLAM.

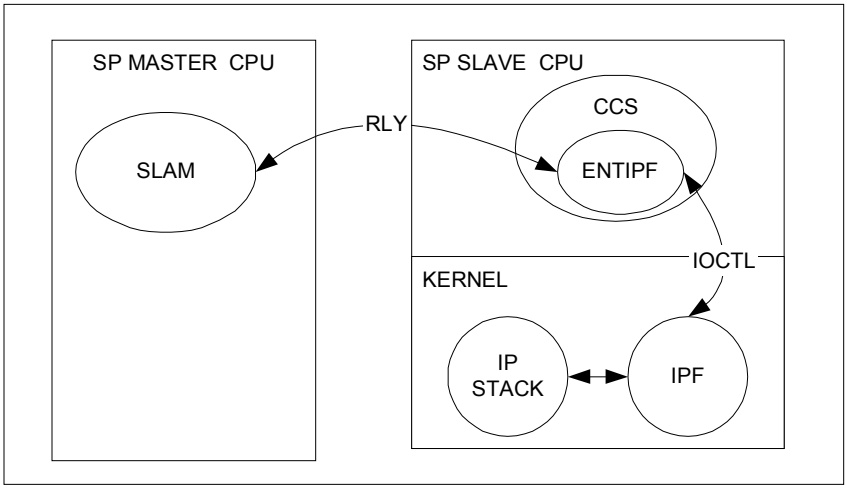


Figure 22

10.16.2 SLAM Responsibilities for IPF

- When TL1 command is entered with a new destination port and protocol, assign a CCS. Then send the port and ccsId(slot and CPU) to the ENTIPF to be programmed into the IPF.
- If a Delete command is entered, the port/ccsId must be deleted from the IPF.
- If the TL1 command changes the port, then a Delete followed by a new Create message is sent to the IPF. (Modify message will not work in this case).
- Rewind from database the port configuration and send a bulk message to the ENTIPF. This will be the same as an audit where every entry is new for the IPF.
- When a CM is failed over, a special fail over message is sent to the CM with just the from slot and the to slot.

- On an SP failover, the new active will perform an audit with the IPF. This involves sending a bulk audit message with all the existing ports/ccsId pairs. This is the same as a rewind. The response back will contain info like how many failed to be programmed and how many were stale. These will be used for debugging.
- The SLAM must be able to request and display (via TL1 or logging) the IPF stats.
- Calls to the IPF must be non-blocking.

10.16.3 ENTIPF Interface

ENTIPF is the relay entity that will reside in the CCS. This is just a small thread that will interface between the SLAM and IPF. The interface to the SLAM will be a loosely coupled PST RELAY message with the same structures as defined in Reference 5) 79-3322, MGC IP Forwarding Functional Specification, D. Dafoe. That way the ENTIPF will just strip off the PST header and stick the address of the IPF message right into the IOCTL. The IOCTL can block ENTIPF until we get a response. The response can be then be removed from the IOCTL and added right back into the reply Relay message back to the SLAM.

10.16.4 SLAM-ENTIPF Create

The Create command can program a destination port to a ccsId. Although this can be a bulk command, it probably will be used just for individual creates from the command line input. SLAM will need to check the return value for errors and alert the user.

10.16.5 SLAM-ENTIPF Modify

This command will allow the ccsId to change for an existing destination port (key). This command is used when an MG is moving to an a different CM. If a user modifies a port for an existing MG or other protocol, the SLAM will have to look up the old port for this MG, delete the old one then create a new one. Again bulking is supported and can be used when we support moving a CM's worth of CCS's to a different CM. LAM will need to check the return value for errors and alert the user.

10.16.6 SLAM-ENTIPF Delete

The delete command will be used when a port is changing or a protocol or MG is deleted. Just a port is passed in to be deleted. SLAM will need to check the return value for errors and alert the user.

10.16.7 SLAM-ENTIPF Audit/Rewind

The audit command is used during a rewind or after a failover of an SP. The audit command has flags for each entry for START, ENTRY, END. These can also be bulked up where the first entry of the first packet will have the START flag. Multiple packets can be sent as long as each entry has the ENTRY flag set. The last entry of the last packet must have the END flag set. For each packet sent to the IPF, a response will be returned with a status for each entry. On the last packet with the END flag, the number of Stale entries and number of discrepancies will be returned. For a rewind, the number of Stale should be 0.

10.16.8 SLAM-ENTIPF Failover

The failover command is used when a CM fails over to the Standby CM. When this happens the IPF needs to change the slot number for each entry that was on the old Active CM. For this case, a FROM_SLOT and a TO_SLOT will be sent in to the IPF. The return should be just a SUCCESS or FAIL.

10.16.9 SLAM-ENTIPF Drv Stats

This command returns the number of successful IP forwarded packets and the number of failed ones with an array of info for the failed lookups. The theory here is that the array is a sniffer for 'failed to forward' packets. The port and source IP address of packets with a destination port in the valid range (see 10.12) that do not match anything programmed in the IPF, are stored in the array. The array will wrap if not read out. The count (numFailedLookups) is the number in the array. This count and the array are cleared after every read.

The SLAM can request this data from a TL1 RTRV command, a sigdbg cmd or in a polling loop. The output will be dumped to the logs or TL1 screen. This is used to find miss-provisioned ports. If the wrong port was entered in the SIP phone or MG, then it would not be forwarded and this would flag it.

Optionally the SLAM or ENTIPF could periodically read this array and send an event to FAM if a threshold is crossed for one port/srcIP pair.

10.16.10 SLAM-ENTIPF Port Stats

The port stats is stored in the IPF. The SLAM can request the data on one or many ports. The return structure contains an the number of successful and failed lookups for each port passed in. This is a bulk message so the stats of many ports can be retrieved at once.

This has many possibilities for debugging. The SLAM can send in all the programmed ports and get a response for each port. This can be used to see what protocols are being passed to the CMs. The port stats requested can be of any value, so if all ports are scanned (1-64K) then the SLAM/user could check for miss provisioning.

> ~~**Issue 13:**~~**Issue 7:** Are these counters cleared after reading, do they wrap or max out? *Answer: These wrap.*

10.16.11 SLAM-ENTIPF Destination Port Range

The valid range of destination ports needs to be setup. There is a default range, but this can be changed on the fly. The purpose of the range is to capture stats in this range. This is mainly for debugging purposes. Changing this will effect the reporting of the driver and port stats.
This is write only IOCTL that will take in the start and end port values. The SLAM must keep the value in persistent memory and will need to programmed at startup.

~~Issue 14:~~~~Issue 8:~~ Should the IPF reject port creates when the port is outside the valid range.
Answer: No, this config is only for showing stats and not critical error. Just not see stats on that port. This is checked in the SLAM prior to sending.

10.17 Call Capture and Trace call

The SLAM will need to handle Call capture and trace call for the MGC. A call can be on any CCS/MG. The SLAM will have to collect the traces.

This has not yet been implemented. This will be updated when that task is scheduled

10.18 sigdbg

The present sigdbg command is in the form of sigdbg:::::"string". The string is decoded in tsmcSigDebugCmd() in tsmc_sigdbg.c. For the MGC, this needs to be forwarded to either the PLM, all CCS, one CCS, Router, or others. The command will now come to the SLAM instead of PLM. The SLAM will parse it out and forward it to whoever wants it.

Proposed sigdbg format:

sigdbg:::::key ['help'] ['mgname'] [cmd string]

The 'help' string will return a handy comment that can tell the user some useful information. The command will not be forwarded to the function in this case. The 'mgname' will be used to forward to a specific CCS. If it is not present, and command is destined for a CCS, then the SLAM will broadcast it to all CCSs.

A new array of structures will be deployed that will facilitate the forwarding and allow for better features.

```
typedef struct sigDbg
{
       char   *key;
       char   *helpStr;
       U32    *func;
} SigDbg;

struct SigDbg sigdbgTbl[]=
{
       {"rum", "Used to debug the Router Update Manager",rumDbg},
       {"ccdbg", "Set debug level, default=0x100FFFF",ccDbg},
       {"ccrm", "Debug support for ccrm for the MG",fwdToPlm}
}
```

The big if/else part of the existing sigdbg can be moved to this format. Functions will be created for each key decode. The function can forward the whole sigdbg command to another application like PLM or a CLAM.

A ‘for’ loop can walk through the array and compare the key from the first entry in the string. When a key is found followed by ‘help’, then the helpStr will be returned. When only one compare is found (with no help), then the string will be passed to the function. If there are multiple partial matches, maybe all the helpStr’s for the partial matches are returned to tell the user what can be used.

10.19 Miscellaneous SLAM Stuff

There are multiple points that need to be captured in the spec. They are:

- When a CM comes in service, the SLAM needs to provision the IP Forwarder as well as the CCSs on the CM. If the IP forwarder was already programmed (CM was in then pulled out then put back in), a port audit needs to be done for the CCSs on that CM.
- SLAM needs to know the status of the GDI on each CM. This is used to declare if the CM is in sync or not.
- When a CCS goes down, the SLAM needs to broadcast a message to all CCSs and XUs to handle the event. The CCSs must forward this to the BICC layer to tear down calls if there is no failover.
- The TL1 agent will now supply a transaction ID with each command. The SLAM needs to keep track of this in it’s FSM.
- Voice Mail support. The old PLM will be running on the SP. This will keep the voice mail support (VMS). But, the CMS Circuit ID Table needs to go to every CCS.
- DNS support. MG's, SIP Feature servers and MGCP equipment can be configured on the MGC with either an IP address or a FQDN. The CCS only knows about the IP address, so if a FQDN is provisioned, the IP address is discovered with DNS lookup, then IP address is sent to CCS. The CCS can not run DNS lookup directly. The FQDN is stored in timesTen so when ever the CCS comes up, a DNS lookup must be performed. Periodically the DNS lookup must be run and if the IP address changes, then the CCS must be notified via a SLAM/CLAM interface. This should not have an impact on the IP forwarder...unless we have to go to forwarding on the source IP address (not present design). We will need a process on the SLAM where these FQDN's are registered and supports this periodic updates.

10.20 Questions/Issues

10.21 Testing

11 CLAM DESIGN

11.1 Block Diagram

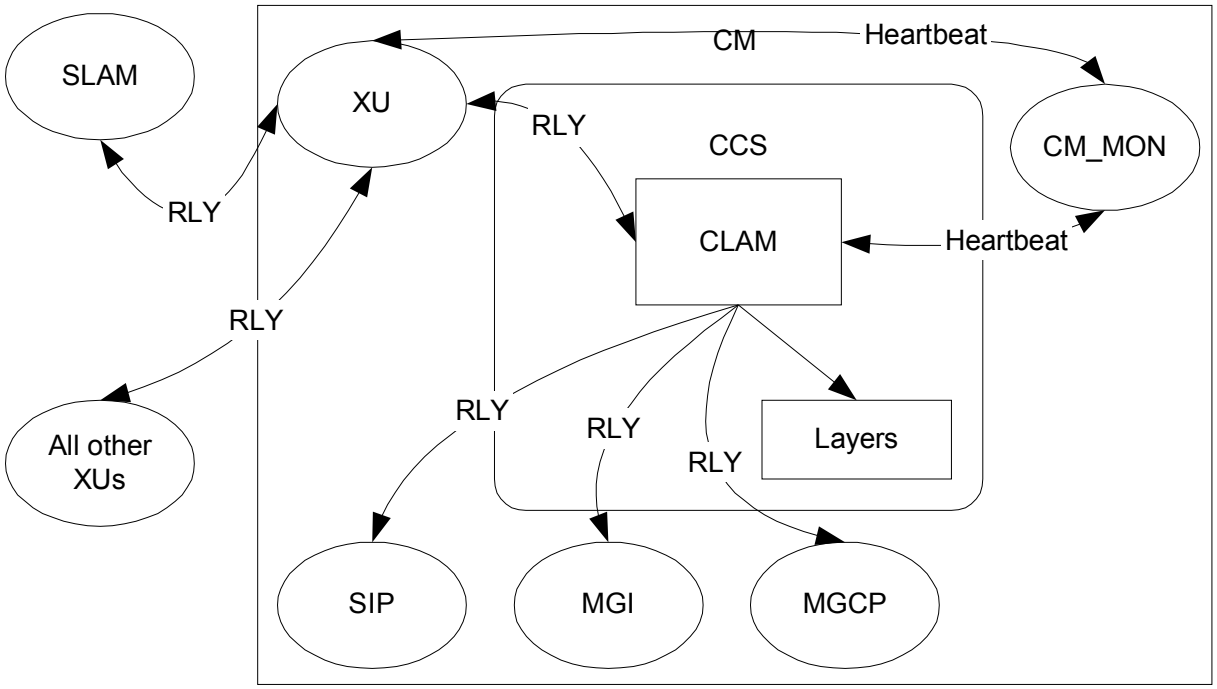


Figure 23

11.2 API Definition

11.2.1 MGI

Interface to MGI is defined in MGC MEGACO Support Library Functional Design Specification (MMSL). A module is required to handle the Board Service changes defined in 79-3326, PLM/CLAM ServiceChange Handling, M. Pralat. The new module is currently a PLM module called PLM_MMSL.

11.2.2 SIP

Same interface as in earlier designs (3.8 and 3.9).

11.2.3 MGCP

Same interface as in earlier designs (3.8 and 3.9).

11.2.4 CM_MON

Interface is defined in 79-3311 (Compute Module Monitor Software Specification).

11.2.5 SLAM

This interface is defined in the SLAM interface with CLAM. This will include a new API with data being sent from the SLAM. The data will include all the timesTen data that is required to provision the appropriate layer. A message queue will be used to store up multiple commands. The CLAM will handle one at a time and reply when completed. Then get the next message.

11.3 CLAM initialization

CLAM initialization consists of stripped down version of existing module initialization routines for TSMC and PLM. The clamActvInit function will have the init code for all the CLAM modules. The main steps in the initialization routine are as follows:

1. Timer Queue initialization
2. CLAM data structures initialization
3. Port State Table, Termination Id Table init
4. CLAM internal hash lists initialization for the PLM based modules. FSM matrices for the modules that are handled by CLAM will be initialized.
5. The watchdog thread is created for sanity punching of the layer threads in CCS.
6. Initialize CLAM Config FSM (clamFsmCb)
7. Start the CLAM config FSM (genCfg relay, setup relay channels, setup CLAM FTHA (message router, system manager))

11.4 CLAM States and Data Structures

These states should match the states in CCS states

```
enum
{
    CCS_OOS=0,
    CCS_IDLE,
    CCS_BOOTING_ACT,
    CCS_BOOTING_STBY,
    CCS_ACTIVE_STANDALONE,
    CCS_ACTIVE_SYNCING,
    CCS_ACTIVE_SYNCED,
    CCS_STANDBY_READY,
    CCS_STANDBY_SYNCING,
    CCS_STANDBY_HOT,
    CCS_MAX_STATE
};
```

There is a single instance of CLAM Control Block. The following fields are identified.

```
typedef struct clamFsmCb
{
    U8 clamType;                /* to be able to tell if its a CLAM on CCM */
    U8 ccsState;                /* one of the above CCS states */
    U8 stbyReady;               /* TRUE: ready, FALSE: not ready
                                   (relevant to active CLAM) */
    U8 cfgFsmState;             /* internal config FSM state */
    U8 rxDwnldSeqNo;            /* received playback request */
    U8 txDwnldSeqNo;            /* acknowledged playback request */
    U8 mgName[MAX_MG_NAME];     /* this is '\0' if no MG associated */
    U8 relayState[MAX_RLY_ENT]; /* TRUE:up, FALSE:down */
    /* other stuff .. */
} ClamFsmCb;
```

```
cfgFsmState – is the internal CLAM state to keep track of the internal events
affecting config FSM on CM.
```

relayState – This array maintains state of relay between CLAM and each of the processes in CCS suite, that is MGCP, SIP, MGI, XU. Its TRUE when relay is up and FALSE when relay is down. The cfgFsmState is WAIT_FOR_LOCAL_RLY when its still waiting for all the relays to be up.

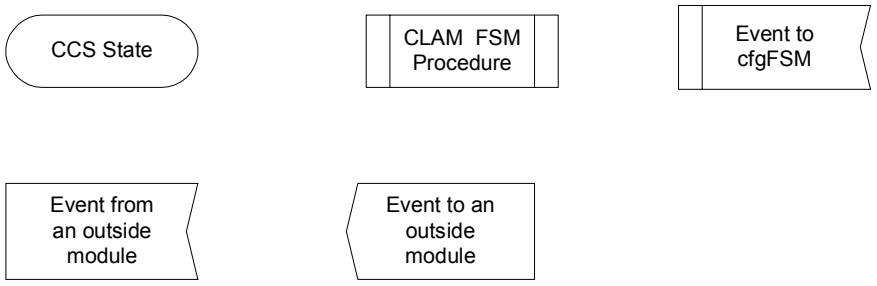
The CCS state change is handled by the stateChangeHandler() callback handler. This function is called whenever CLAM receives CCS state change indication message from SLAM. Later if the CCS state change initiation is moved to some entity other than SLAM the callback can be exposed to the new entity.

typedef U32 (*stateChangeHandler)(int oldState, int newState);

11.5 Boot, Replication, and Failover FSM

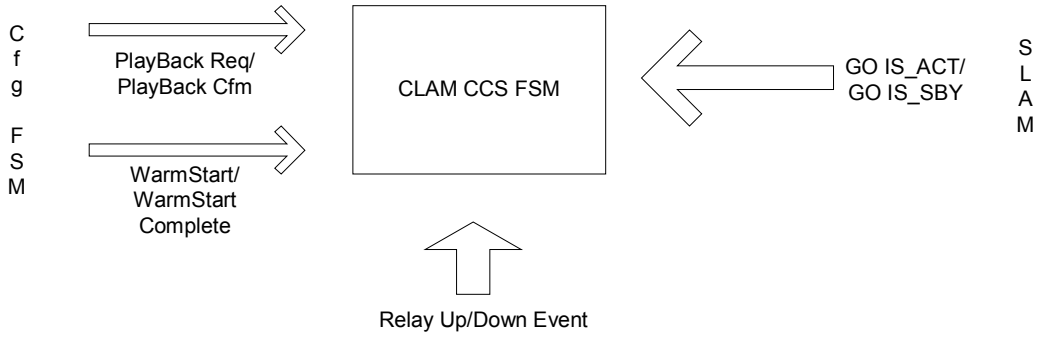
This state machine will handle the transition of the states defined in CLAM States and Data Structures.

11.5.1 SDL Representation



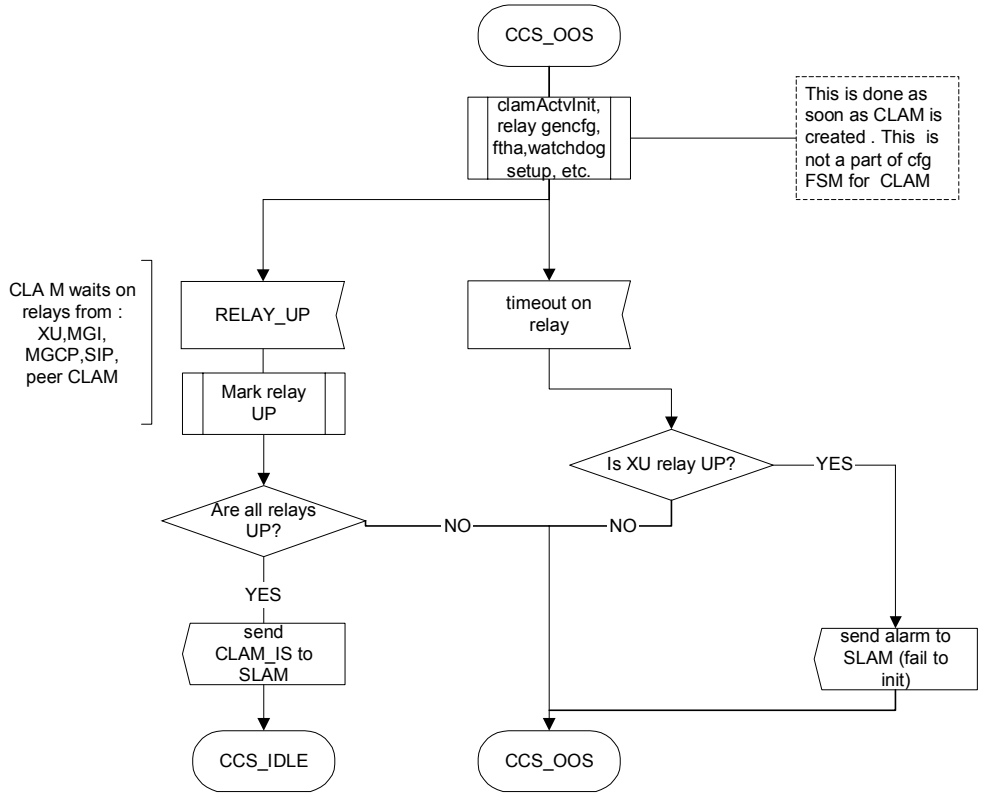
11.5.2 FSM Events

The FSM events listed below are depicted in the subsequent CCS state FSMs.

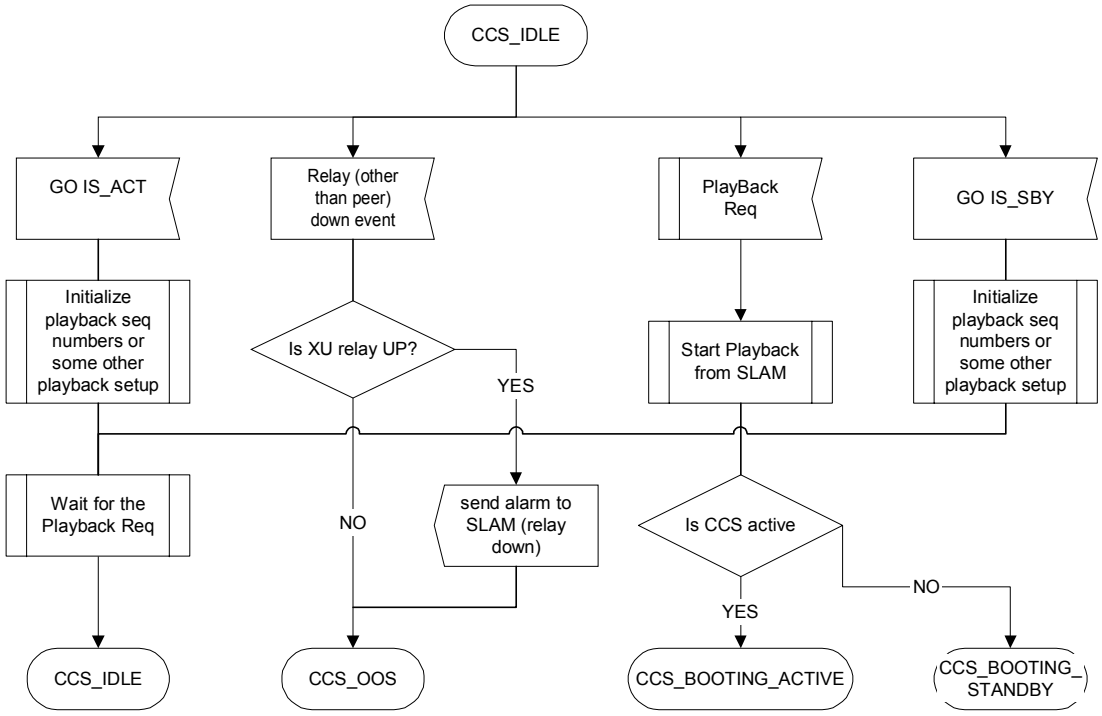


11.5.3 CCS States

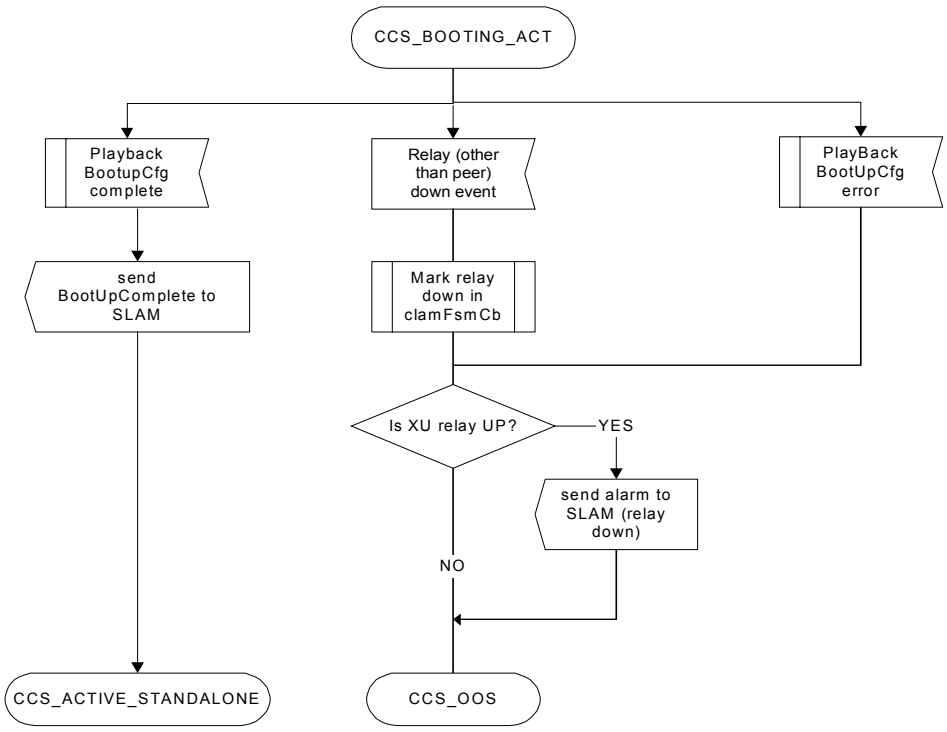
11.5.3.1 CCS_OOS



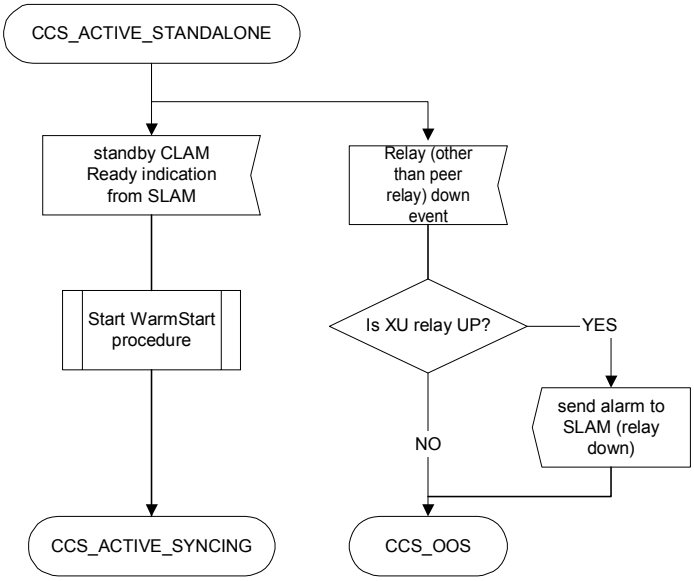
11.5.3.2 CCS_IDLE



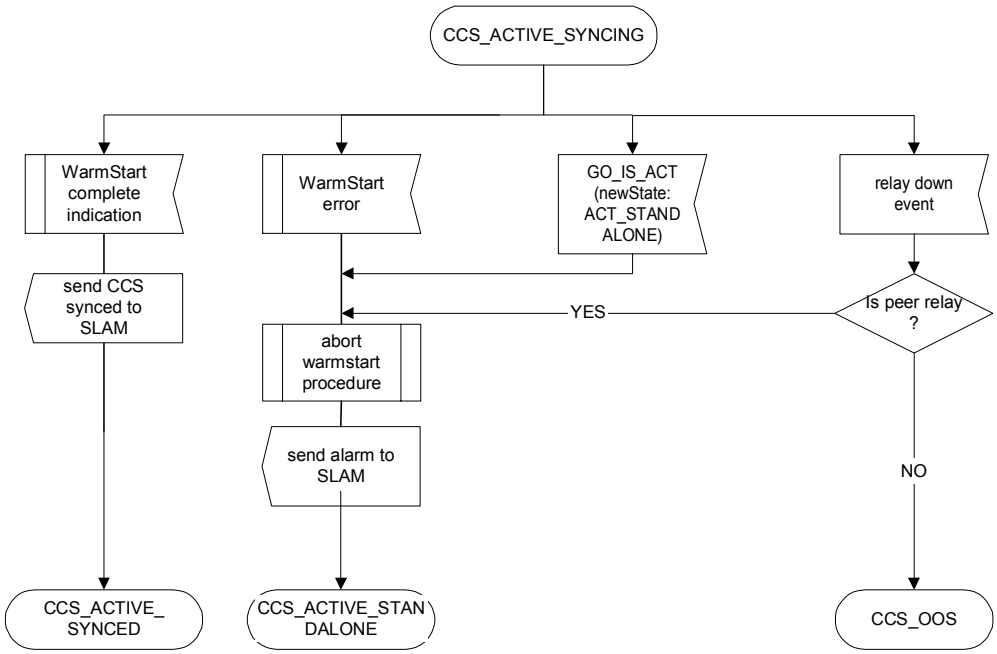
11.5.3.3 CCS_BOOTING_ACT



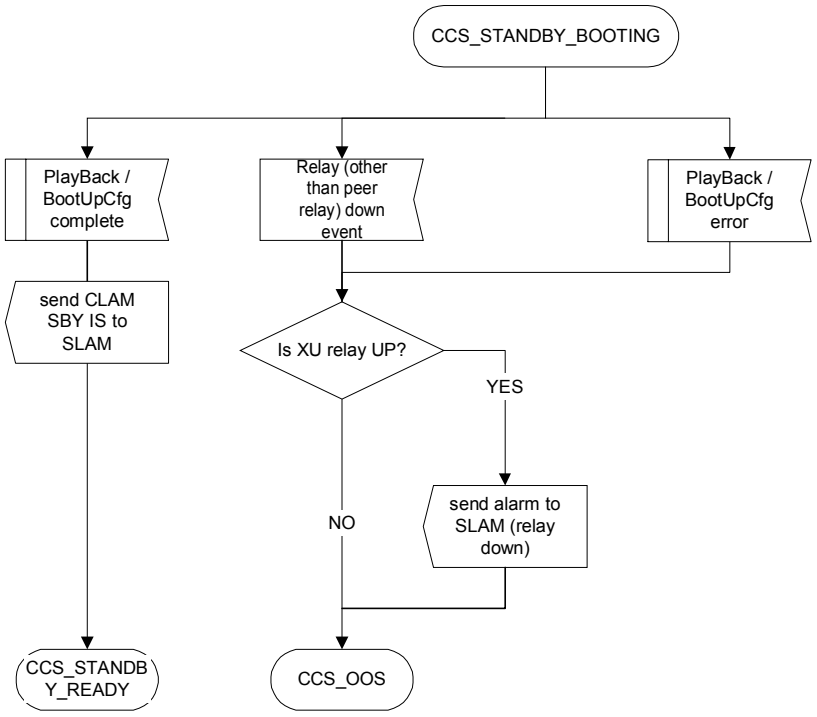
11.5.3.4 CCS_ACTIVE_STANDALONE



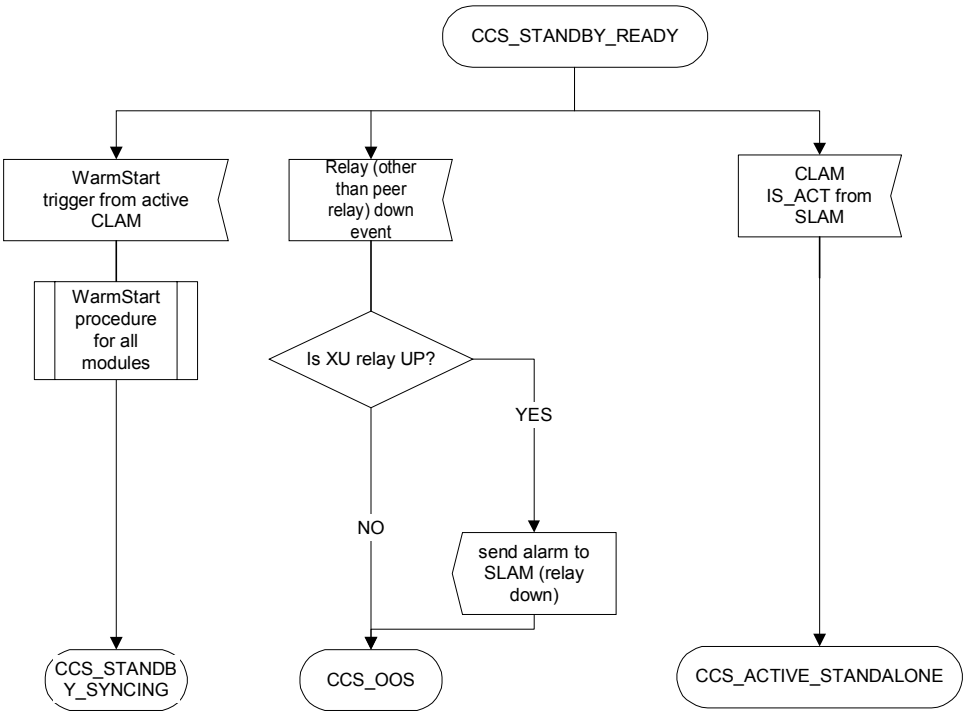
11.5.3.5 CCS_ACTIVE_SYNCING



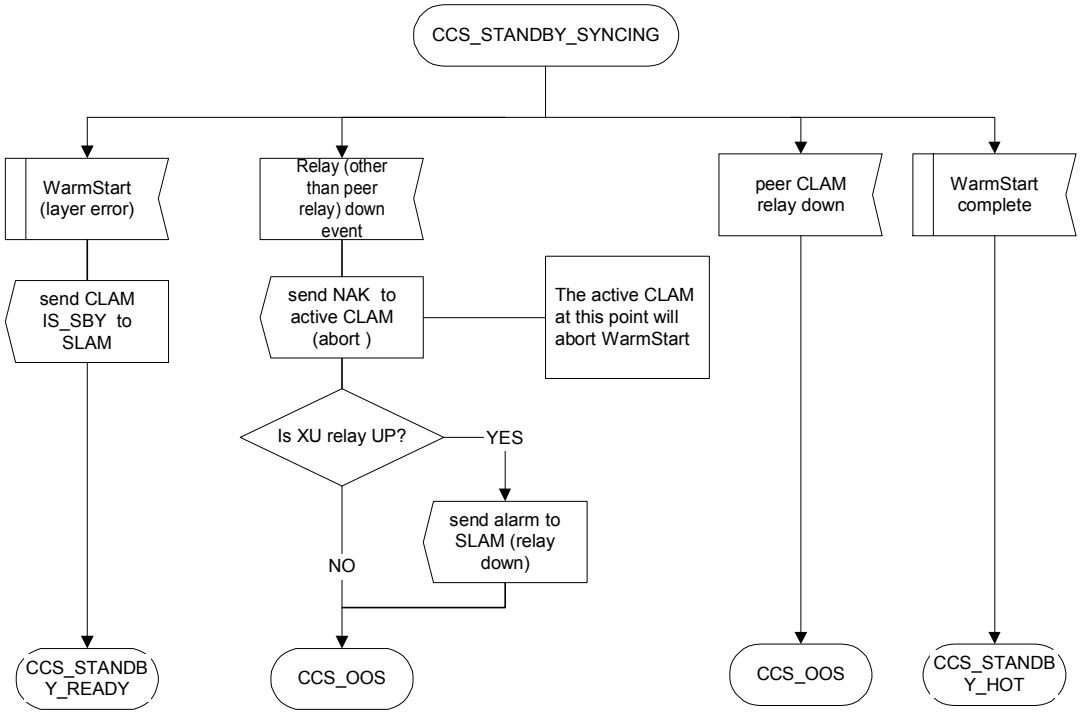
11.5.3.6 CCS_STANDBY_BOOTING



11.5.3.7 CCS_STANDBY_READY



11.5.3.8 CCS_STANDBY_SYNCING

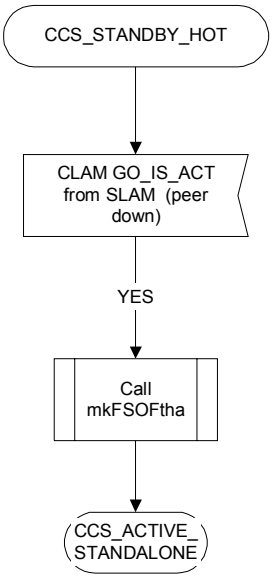


Note 1: A general note for all the FSMs. The FSM will wait for relay up events depending on the type of CM (the one on CM or CCM). This is for the possibility that the CCM may not have all the processes in CCS suite.

11.5.4 CCS Failover

The CCS failover can be initiated by SLAM. CLAM will report any alarm condition that is received from layer in CCS or if the relay is lost (USTA event). The standby CLAM when in CCS_STANDBY_HOT state will initiate the forced switchover FTHA procedure.

If a TL1 transaction was in progress during the CCS switchover (controlled switchover case) then will SLAM wait for the whole command to be completed.



11.5.5 CLAM Replication

The CLAM replication is done under following scenarios:

1. Service Change events received from MGI. For the Plexus MG these events will be mapped to the Board State Change and Port State Change events in PLM. The replication is done over relay and existing scheme of sequence numbers can be reused.

2. After the CCS failover the Service Change Audit will be initiated. (Who initiates this audit? CLAM will ask MGI to initiate or MGI does independently as soon as CM becomes active). The correspondingly generated Board and Port State Change events on the active will be replicated to the standby.

3. TL1 commands are replicated to the standby if a layer needs to be configured (ED/ENT) commands. This will follow the existing scheme.

11.5.6 Service Change Handling

CLAM will process the Service Change Requests at the port level (first release). The Service Change event is a newly defined event.

The MMSL layer receives the ServiceChange message from the MG. MMSL will pass it to the CLAM layer as an unsolicited status indication (EVTLMMSLSTAIND). The event is currently handled by the PLM_MMSL entity added to the existing PLM layer. PLM_MMSL decodes the message since the USTA messages received from MMSL may be: ServiceChange, AuditReplies, Alarms, or ServiceChange Replies. When the message is decoded, further action may be necessary, e.g. raise an event to other modules, update state information.

11.5.6.1 Service Change Event

11.5.6.1.1 PLM Event

The new event can be defined as PLM_TERMID_EVT. (why can't these be normal port control events????). This event is handled only on the active CM. The state change/addition/ deletion events will be replicated to the standby.

11.5.6.1.2 PLM Event Matrix

CLAM will receive the newly defined Service Change Event. The termination Ids can be associated with a unique MGVariant. The MG variant is currently a dynamic property. A new matrix is defined to process the PLM modules that are affected by the service change of the terminationId. To define the matrix to handle the new service change event the following is needed:

TBD – item missing, probably the key.

This is the key that is checked for the validity whenever an event comes into PLM. If the new port event doesn't match any of the keys in the matrix then PLM fails to build a new event for it and hence dumps it on the floor. The MG originated Service Change events have a unique MGVariant allocated at the time of the ent-mg-assoc command. Since these are dynamic in nature the can be single dimensional key which new event type. The key is PLM_TERM_ID_EVT.

The PLM modules currently affected by the Port matrix will be populated in the Matrix definition of the service change event matrix.

11.5.6.1.3 Event Handling

The service change event is handled by mapping this to a unique entry in the PortStateTable. The mapping function can be defined in the Matrix processing functions (preprocess). For this to happen it should be ensured that the port specified in the MMSL request doesn't cross the boundaries for the port table. Once a port, slot pair is generated, these two are passed in the new event to each of the handling modules.

TBD – update this section. Last discussion had replacing the PortStateTb completely with the terminationID tree.

11.5.6.2 Termination Id Creation and Deletion

The termination Id properties are not stored in the persistent memory. These terminations need to be re generated whenever CCS has booted up. The existing scheme of using the TL1Event Queue can be extended for the new terminations as well once the mapping is done.
TBD – what are the properties?
The PLM_MMSL module needs access to the terminationID tree (MG context, and tree context could be in a shared global data structure). The USTA arrives from MMSL and PLM_MMSL needs to look up the terminationID in the terminationID tree. Expect that ent-mgc-mgassoc will create the MGVariant and terminationID tree. Other TL1 commands (CAS, ISDN) will add terminationIDs to the terminationID tree.

During Boot Up:
The CLAM config FSM needs to perform the audit with MG to get the ports data before the tl1 playback can take place. (Or is SLAM going to provide this data on bootup before the module playback takes place.???).

After the CCS failover:
The CLAM needs to initiate and audit with MG to get the service change events. Once this audit is done the standby and active can proceed to the internal data sync up.

11.6 Modifications from PLM

This section will detail the changes required for the CLAM. Most of the code is a conversion from the existing PLM modules. The plm_module.[ch] files will be copied to signaling/lam/clam. The main deviations from the existing PLM are as follows:

- Port handling/ds0 handling – PLM needs to have the processing capability for the multi level service change request from MGI.
- The existing matrices defined for the processing port state change events can't be directly used in the existing PLM code. The new matrix needs to be defined to be able to feed these status change/create/delete events to the existing PLM modules.

- The new matrix definitions will be required to process the ds0/channel level state changes and audits.

- Existing Config FSM in TSMC on SP gets the port audit messages in a controlled fashion. That is eqm sends the port state changes using the DS1_PORT_AUDIT_START and DS1_PORT_AUDIT_END messages. This way the port audit is a part of tsmc config fsm.

  Since ports/channels are remotely terminated on a MG the audit needs to be initiated from entity on CM after the end of CLAM config FSM. This can be implemented as Port/Channel Audit modules in PLM modules like CAS, ISDN IF, ISDN LINK, IUA LINK, GR-303 Channel. Each module will be added to the event matrix handling the Service Change event.

- PLM Core module will have the TL1 command input queue. This will be a linked list of command requests from SLAM. Routines to traverse and de-queue the commands one by one will be new addition to existing PLM core.

- TL1 Command concurrency matrix needs to be added. This will be static in nature that is, its going to be populated at compile time and will list the module dependencies for each command matrix. Each time a TL1 event is de-queued from TL1 command input Q command concurrency matrix is referenced to check if any of the dependent module listed in the concurrency matrix is currently being processed by PLM. If yes then this module will be retained in the TL1 input Q. If it is independent of all the currently active PLM modules then it is de-queued and processed.

  The matrix is maintained on per PLM module basis having a list of modules that can get affected once the current new PLM module is processed for a given PLM TL1 event.

- The FTHA configuration will be modified to remove IOM handling.

11.7 Command FSM

This FSM is required to handle each command that comes from the SLAM. This is done using the existing PLM Matrix and module FSM support.

11.8 Miscellaneous CLAM Stuff

There are multiple points that need to be captured in the spec. They are:

- There are 2 OOS modes (Forced and Normal) for an MG Association. The CLAM will need to send this info to multiple layers. Forced means to disconnect all calls ASAP. Normal is that no new calls are allowed, and old calls will be released when users hang up. Also need to block all the CICs associated with the MG. Once all this is done, then need to send a message to the MG via MEGACO that the MG is OOS. ??? .
- The CLAM on the master CCS on a CM needs to tell GDI on CPU1 that it can 'go-active'.

12 Glossary

CCM	Centralized Compute Module
CCS	Call Control Signaling
CLAM	CCS Layer and Alarm Manager
CM	Compute Module
EMF	
EQM	
IPF	IP Forwarder
SLAM	System Layer and Alarm Manager

13 Appendix A: TSMC Functions

The TSMC functions that exist in 3.8 today are to follow. These features are required in the new architecture where there exists a SLAM and a CLAM. Some items are in TSMC style code and may need to be moved to the PLM design to support multi-session TL1 commands.

1. **PLM UNSOLICITED EVENTS (ENTMCA)**
    i. Alarm handling entry point for PLM.

2. GoAhead interface (ENTGIT). These include the following:
    a. EQP_CLASS_INT
        i. This handles the board events from the EQM.
    b. SM_VOA_CMD_CLASS_INT
        i. Voice over ATM support.
    c. SM_CMD_CLASS_INT
        i. TL1 Provisioning
            1. SM_ENT_REQUEST
            2. SM_ED_REQUEST
            3. SM_DLT_REQUEST
            4. SM_RTRV_REQUEST
        ii. TL1 Agent API for bulk requests
            1. SM_GETNEXT_REQUEST
            2. SM_GETMULTI_REQUEST
            3. SM_GETNEXTMULTI_REQUEST
        iii. Signal capture support for LinkSets, ISDN and CAS
            1. STA_SIGCAPT_CMD
            2. STP_SIGCAPT_CMD
        iv. GR303 support
            1. SM_SWTOPROTN_REQUEST
            2. SM_OPR_REQUEST
            3. SM_RLS_REQUEST
        v. SIG_DEBUG_CMD
            1. Supports all sigdbg commands. No need to move to PLM.

3. Peer TSMC messages (TSMC_CLASS_INIT)
    a. TSMC_PEER_BRD_UPD_REQ:
4. This function handles a board state table update request from the active
    a. TSMC_PEER_BRD_UPD_CFM:
5. Handle board update confirm from standby.
    a. TSMC_PEER_IOM_PROT_SW_REPL_REQ:
        i. Update FTHA router and system manager with new pProcId.
    b. Handle standby replication states
        i. TSMC_PEER_IOM_PROT_SW_REPL_CFM:
        ii. TSMC_PEER_READY:
        iii. TSMC_PEER_STANDBY_T1_AUDIT_DONE:

iv. TSMC_PEER_STANDBY_T1_OOS_SYNC_DONE:
v. TSMC_PEER_CFGFSM_DONE:
vi. TSMC_PEER_START_T1_OOS_SYNC:
vii. TSMC_PEER_GENCFG_DONE:

c. TSMC_CHASSIS_UP_TICK
   i. Update sysUpTime

6. SIG_REPL_CMD_CLASS_INT:
   a. Receiving T1 Replication events
   b. TSMC_SIG_REPL_CMD_AUDIT:
      i. Standby receiving T1 port status audit.
   c. TSMC_SIG_REPL_REQ:
      i. Standby receives T1 events processed
   d. TSMC_SIG_REPL_ACK:
      i. Positive response from standby for TSMC_SIG_REPL_REQ
   e. TSMC_SIG_REPL_NAK:
      i. Negative response from standby for TSMC_SIG_REPL_REQ

7. T1 driver Messages (SIG_CMD_CLASS_INT)
   a. T1 Audit messages from
      i. DS1_SIG_PORT_AUDIT_START:
      ii. DS1_SIG_PORT_AUDIT_END:
   b. Handle OC port create/delete query/delete event from the oc_hm.
      i. OC_SIG_PORT_CREATED:
      ii. OC_SIG_PORT_CREATED_MULTI:
      iii. OC_SIG_PORT_DELETED:
      iv. OC_SIG_PORT_DELETE_QUERY:
      v. OC_SIG_PORT_AUDIT_START:
      vi. OC_SIG_PORT_AUDIT_END:
      vii. ATMT3_SIG_PORT_CREATED:
      viii. ATMT3_SIG_PORT_CREATED_MULTI:
      ix. ATMT3_SIG_PORT_DELETED:
      x. ATMT3_SIG_PORT_DELETE_QUERY:
      xi. ATMT3_SIG_PORT_AUDIT_START:
      xii. ATMT3_SIG_PORT_AUDIT_END:
      xiii. OC_SIG_PORT_STATE_CHANGE:
      xiv. OC_SIG_PORT_STATE_CHANGE_MULTI:

8. Message from active GIT (Go Active) (TSMC_CONTROL_CLASS_INT). These are control messages from the other TSMC (either active or standby). They are used to control the state, syncing and replication of the standby.
   a. Standby control
      i. TSMC_CTRL_INIT_STANDBY:
      ii. TSMC_CTRL_INIT_ACTIVE:
      iii. TSMC_CTRL_START_SYNC:
      iv. TSMC_CTRL_STOP_REPLICATION:

v. TSMC_CTRL_GO_ACTIVE:

b. TSMC_CTRL_KICK_T1_EVENT_QUEUE:
   i. This functions checks for pending T1 events on the queue. If any exist, process them.

9. TSMC_FROM_GIT_CLASS_INT:
10. This routine is the dispatcher for events received from the STANDBY GIT during the synchronization process.

11. NDL_EVT_CLASS_INT:
12. This is the entry point into PLM for system Nodal state changes

13. ENTSM: /* Trillium Stack Manager */
    a. This section forwards response from TSM, to TL1 via GIT.
       i. EVTSMISETRESP:
       ii. EVTSMIVALRESP:
       iii. EVTSMIGETRESP:
       iv. EVTSMIGETNEXTRESP:
    b. EVTSMITRAPIND is used to check for alarms from signaling then handles them appropriately.

14. ENTTSMC:
    a. This is used for TSMC to send itself loosely coupled calls to itself for various reasons. Looks like it does this partly for prevent a thread of hogging the CPU. It will allow other threads to run.
    b. EVTPPTREPREQ:
    c. EVTPPTREPCFM:
    d. TSMC_CONT_STBY_T1_AUDIT:
    e. TSMC_CONT_STBY_T1_AUDIT_OOS:
    f. TSMC_KICK_T1_EVENT_QUEUE:
    g. TSMC_KICK_RY_EVENT_QUEUE:
    h. TSMC_STANDBY_T1_AUDIT_DONE:
    i. TSMC_START_IOM_PROT_SW_FSM:
    j. TSMC_BRD_BULK_UPD_DONE:
    k. TSMC_MAKE_STANDBY:

15. EVTRYREPUP: //RECEIVED REMOTE RELAY CHANNEL