GENIVI®

Software Management

Thursday 4/28 14:00 – 16:00

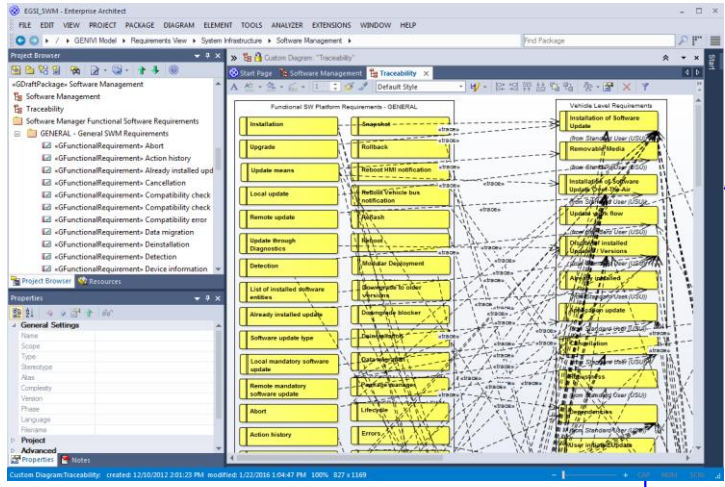
Rudolf J Streif
System Architect, Networking EG Lead
Jaguar Land Rover

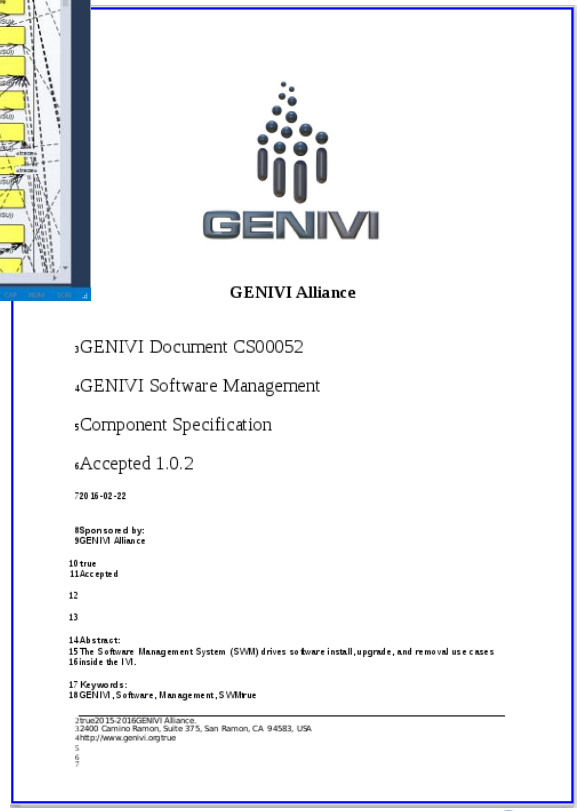
Why Software Management (SWM)?

- SWM is aware of the system context:
  - Postpone software updates when the system is performing critical tasks.
  - Shut down running applications that are being updated.
  - Update, upgrade or migrate persistent data.
- SWM extends beyond the boundaries of the headunit:
  - Update firmware on ECUs
- SWM performs multiple steps in one update:
  - Steps are defined by a manifest.
- SWM provides granularity:
  - Install, update or remove individual software packages.
  - Image entire system partitions
- SWM integrates with the system:
  - Utilize native OS tools such as package management.
  - Integrate with SOTA system.
  - Integrate with GENIVI Node Manager (NSM)

Current Status

- Requirements modeled in EA.
- SWM Component Specification 1.0.2, complete with Franca IDL and use-case diagrams, accepted by SAT in February.
- Proof-of-Concept implementation with simulated and limited actual functionality.





GENIVI Alliance

GENIVI Document CS00052

GENIVI Software Management

Component Specification

Accepted 1.0.2

16-02-22

Sponsored by:
GENIVI Alliance

true
Accepted

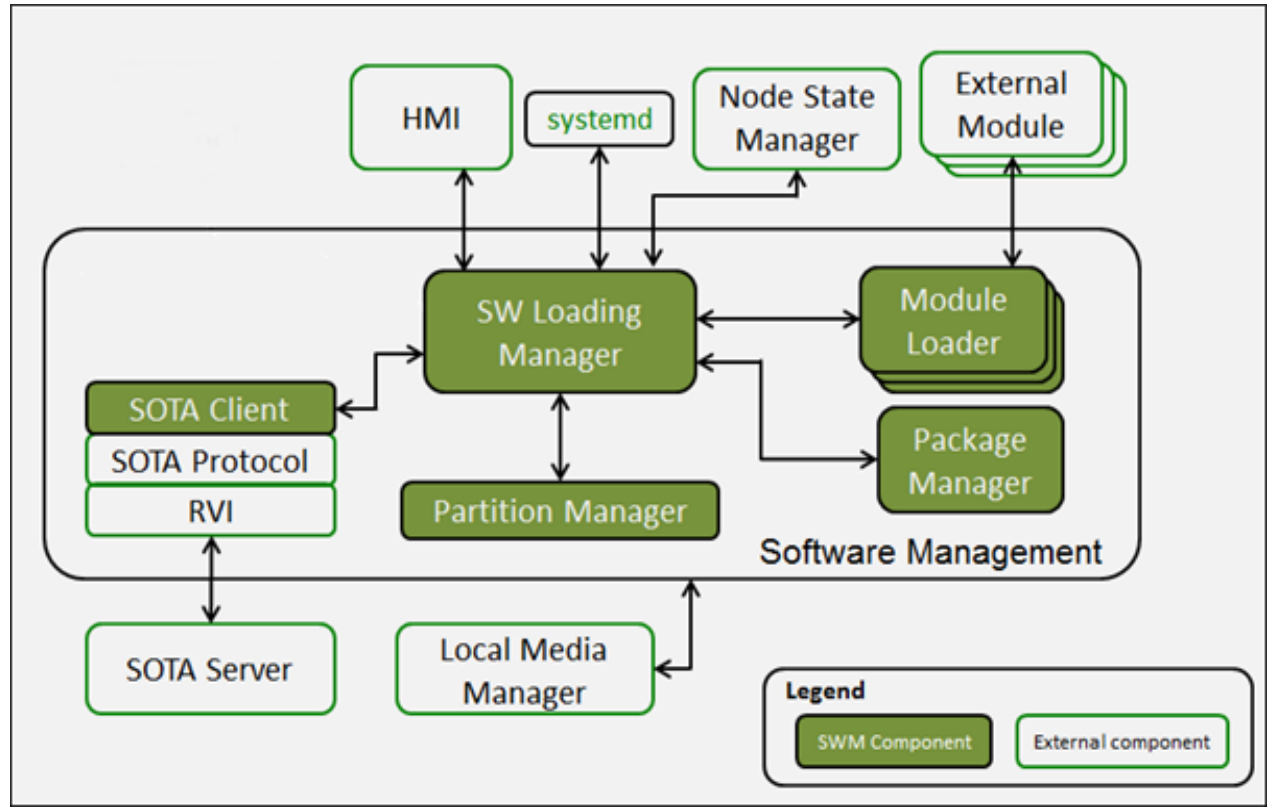
Abstract:
The Software Management System (SWM) drives software install, upgrade, and removal use cases inside the IVI.

Keywords:
GENIVI, Software, Management, SWMtrue

true2015-2016GENIVI Alliance,
2400 Camino Ramon, Suite 375, San Ramon, CA 94583, USA
http://www.genivi.orgtrue

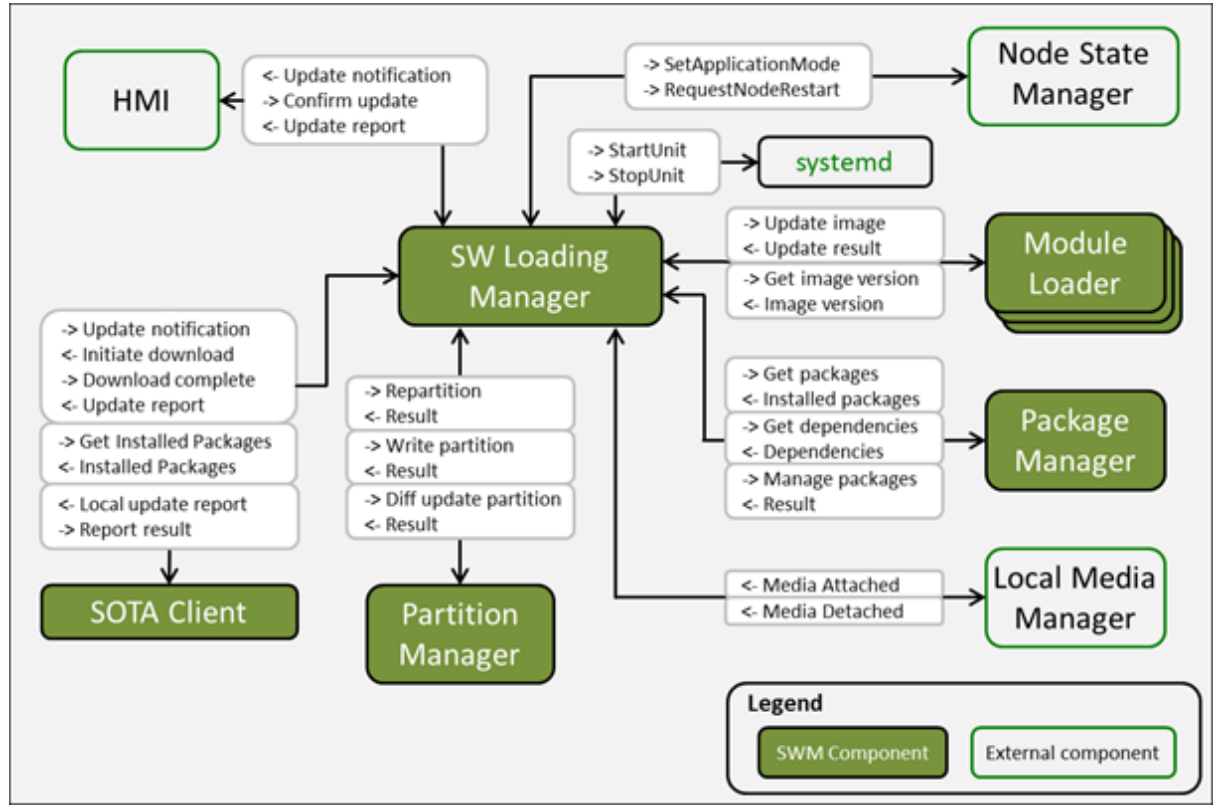
Architecture

- Modular
  - Software Loading Manager
  - SOTA Client
  - Module Loader
  - Package Manager
  - Partition Manager
- Components are separate processes that communicate via dbus messages.
- Software Loading Manager implements the business logic and coordinates software updates through the other components.



Component Interfaces

- Component interfaces are modeled in Franca IDL.
- Component interfaces are implemented as dbus service objects with providing dbus service methods.
- Interface methods can be synchronous (immediate operations) or asynchronous (time-consuming operations).



Software Loading Manager Interface

Command	Description
updateAvailable	Called by SotaC or DiagToolMgr to indicate that an update is available for download
downloadComplete	Called by SotaC or DiagToolMgr to indicate that an update download has completed.
updateConfirmation	Called by HMI to approve or decline the download and installation of an update.
operationResult	Called by PartMgr, ModLdr, or PackMgr to report the result of a software operation bundled in an update.
mediaAttached	Used by LocMedMgr to report to SWLM that a local media (USB stick, DVD, etc) has been connected and mounted.
mediaDetached	Used by LocMedMgr to report to SWLM that a previously connected local media (USB stick, DVD, etc) has been disconnected and unmounted.
getInstalledSoftware	Used by HMI and SotaC to retrieve a list of all installed packages and firmware versions.

SOTA Client Interface

Command	Type	Description	Callback to SWLM
initiateDownload	Async	Used by SWLM, once it has been notified of an available package, to initiate a download of said package.	downloadComplete
abortDownload	Sync	Used by SWLM to abort an ongoing download started by a previous initiate_download call	N/A
updateReport	Sync	Used by SWLM to report the success/failure of software operations bundled in an update.	N/A

HMI Interface

Command	Type	Description	Callback to SWLM
updateNotification	Async	Used by SWLM to forward a package notification received from SotaC. The HMI is expected to automatically confirm / decline the package, or pop a dialog asking the user to do so.	updateConfirmation
manifestStarted	Sync	Invoked by SWLM to notify the HMI (and the user) that the processing of an update has started.	N/A
operationStarted	Sync	Invoked by SWLM to notify the HMI (and the user) that the processing of a software operation inside an update has started.	N/A
updateReport	Sync	Used by SWLM to report the success/failure of a package operation to the user. The same command is also issued to SotaC (see above).	N/A

Partition Manager Interface

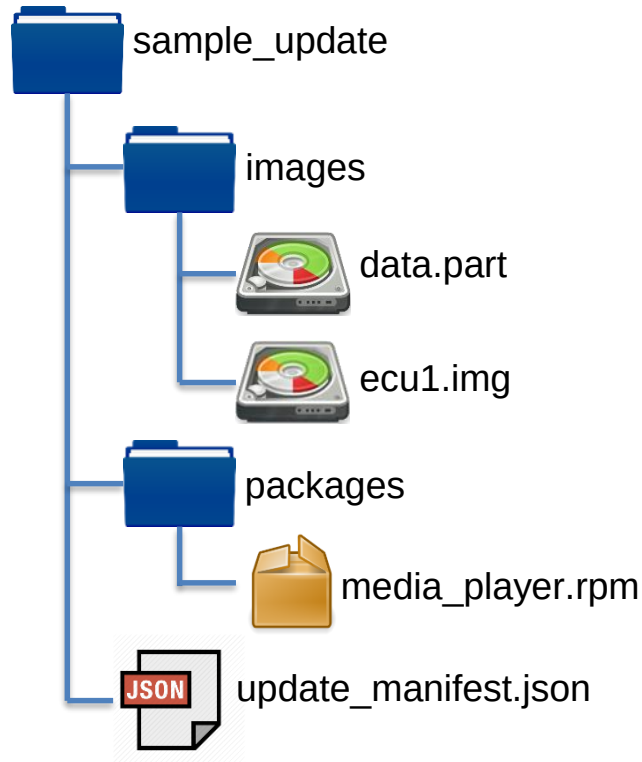
Command	Type	Description	Callback to SWLM
createDiskPartition	Async	Used by SWLM to create a new partition on an unused part of a disk	operationResult
deleteDiskPartition	Async	Used by SWLM to delete an existing partition from a disk	operationResult
resizeDiskPartition	Async	Used by SWLM to move and/or resize an existing disk partition while retaining its content	operationResult
writeDiskPartition	Async	Used by SWLM to write a re-image a complete partition on an IVI disk.	operationResult
patchDiskPartition	Async	Used by SWLM to apply a binary diff to the original content of a partition.	operationResult

Package Manager Interface

Command	Type	Description	Callback to SWLM
installPackage	Async	Used by SWLM to install a new package using the native package manager.	operationResult
upgradePackage	Async	Used by SWLM to upgrade an existing package.	operationResult
removePackage	Async	Used by SWLM to remove an existing package	operationResult
getInstalledPackages	Sync	Used by SWLM to retrieve all currently installed packages	N/A

SWM Update Image

- SWM update images are SquashFS images with an arbitrary directory structure.
- They contain a manifest file in JSON format, named `update_manifest.json`.
- The manifest file contains directives and operations for the software loading manager.
- The manifest references directory and files contained in the update image using relative paths.
- Directories and files can be arbitrarily named as long as SquashFS supports the characters uses.
- Update images can be signed and encrypted if required.



Manifest File

- JSON format with global settings and a list of operations.
- Easily extensible with new fields when new requirements arise.
- Maintains backwards compatibility.
- Currently supported fields and operations are specified by the component specification.
- Some fields are common across operations, others are specific.

```
{
   "updateId": "nano_editor_update",
   "name": "Nano Editor",
   "description": "A small editor so you can hack your infotainment system.",
   "getUserConfirmation": true,
   "showHmiProgress": false,
   "showHmiResult": true,
   "operations": [
      {
         "id": "b9e3865e-9d35-11e5-b235-eb755e788c53",
         "hmiMessage": "Deleting old Nano package.",
         "timeEstimate": 5000,
         "operation": "removePackage",
         "packageId": "nano",
         "onFailure": "continue"
      },
      {
         "id": "d1c96216-9d35-11e5-a00b-57f2b4f03098",
         "hmiMessage": "Installing new Nano package.",
         "timeEstimate": 5000,
         "operation": "installPackage",
         "image": "packages/nano-2.3.6-7.fc22.x86_64.rpm",
         "onFailure": "abort"
      },
      {
         "id": "32c5fe09-e2cc-445d-a903-86932493d063",
         "hmiMessage": "Upgrading Nano package.",
         "timeEstimate": 5000,
         "operation": "upgradePackage",
         "image": "packages/nano-2.3.4-3.fc21.x86_64.rpm",
         "onFailure": "abort",
         "allowDowngrade": true
      }
   ]
}
```

Execution and Exception Handling

- Software Loading Manger executes the operations in the order of the manifest file.
- Operations can define an *on-failure* behavior.
- Status of a software update and its operations are stored in a local database. If an update is interrupted, the Software Loading Manager will continue after the last completed operation.
- Future improvements:
  - Allow definition of arbitrary next operations for *on-success* and *on-failure* (currently on-success continues on to the next operation and on-failure can either continue on the next operation or abort).
  - Allow definition of multiple next operations for on-success and on-failure to provide for parallelism.

Blacklisting and Downgrading

- A SWM update image can contain a blacklist.json file describing packages and images that must not be installed.
- SWM persists the blacklist file and passes the blacklisted packages and images to the Package Manager and Partition Manager respectively.
- Package upgrade operations can explicitly prohibit downgrading of packages. The Package Manager checks the installed version of the package and compares it to the version to be installed.

```
{
    "version": 12239,
    "package": [ "media_player-1.3.4", "backup_camera-3.*" ]
    "firmware": [
      {
        "module_loader": "amplifier",
        "versions”:[ "1.2.*", "1.3.0" ]
      },
      {
        "module_loader": "telematics",
        "versions”:[ "4.2.3-9.?.2" ]
      }
    ]
    "partition": [
        "570123c337570b99f9c77d7a2007bf7bcfda7670a90c9763bc8f96d6edf3a86d",
        "4a694d3b09687ceb57d402feb813635205f617d9295e3539a73e85459d4537eb",
        "54b0e3cfcbece63ca99ab1f3e9365c6146294a6ff92ac100006fe090cde7f78d"
    ]

}
```

Root File System Updates

- The root file system cannot be updated while the kernel has mounted it and it is in use by active user-space applications.
- The SWM specification does not mandate a particular way of updating root file systems as it is dependent on hardware, CPU architecture, bootloader, media layout.
- A variety of approaches exist, to name two:
  - Blue Green Update: The running system installs the new root file system on a secondary partition, copies any configuration and user data, modifies the bootloader entry, and reboots the system using the new partition. Allows rollback to the previous partition but requires twice the media size.
  - Loading Mode: The running system is rebooted into loading mode from a separate partition. It then performs the root file system update on the original partition, restores/copies the configuration and user data, and then reboots the system using the main partition. Rollback not possible but uses less media space.

Proof-of-Concept Implementation

- On GitHub at https://github.com/GENIVI/genivi_swm
- Developed in Python
- Can be run in:
  - Simulation Mode: only log operations
  - Real Mode: perform operations
- Currently only the Package Manager implements real mode and installs RPM or DEB packages dependent on the system.
- Information, debug, warning and error messages are logged to screen and/or file.
- Updates and their operations are stored in a SQLite database.
- Simulated SOTA Client as well as end-to-end integrated with GENIVI SOTA.

Next Steps

- Implementation of real mode for Partition Manager:
  - Using parted for creating, deleting and resizing partitions.
  - Using dd to image partitions that are not root file systems.
  - Investigating Mender (https://github.com/mendersoftware/mender) for root file system image update.
- Implementation of blacklists.
- Integration with Node State Manager.
- Integration with systemd for starting and stopping applications.

Resources

- Confluence Home Page: https://at.projects.genivi.org/wiki/display/SWM/GENIVI+Software+Management+Home
- Component Specification: https://collab.genivi.org/wiki/display/compliance/Software+Loading+Manager
- Tracker: https://at.projects.genivi.org/jira/projects/SWM/summary
- PoC Repository: https://github.com/GENIVI/genivi_swm/blob/master/start_swm.sh

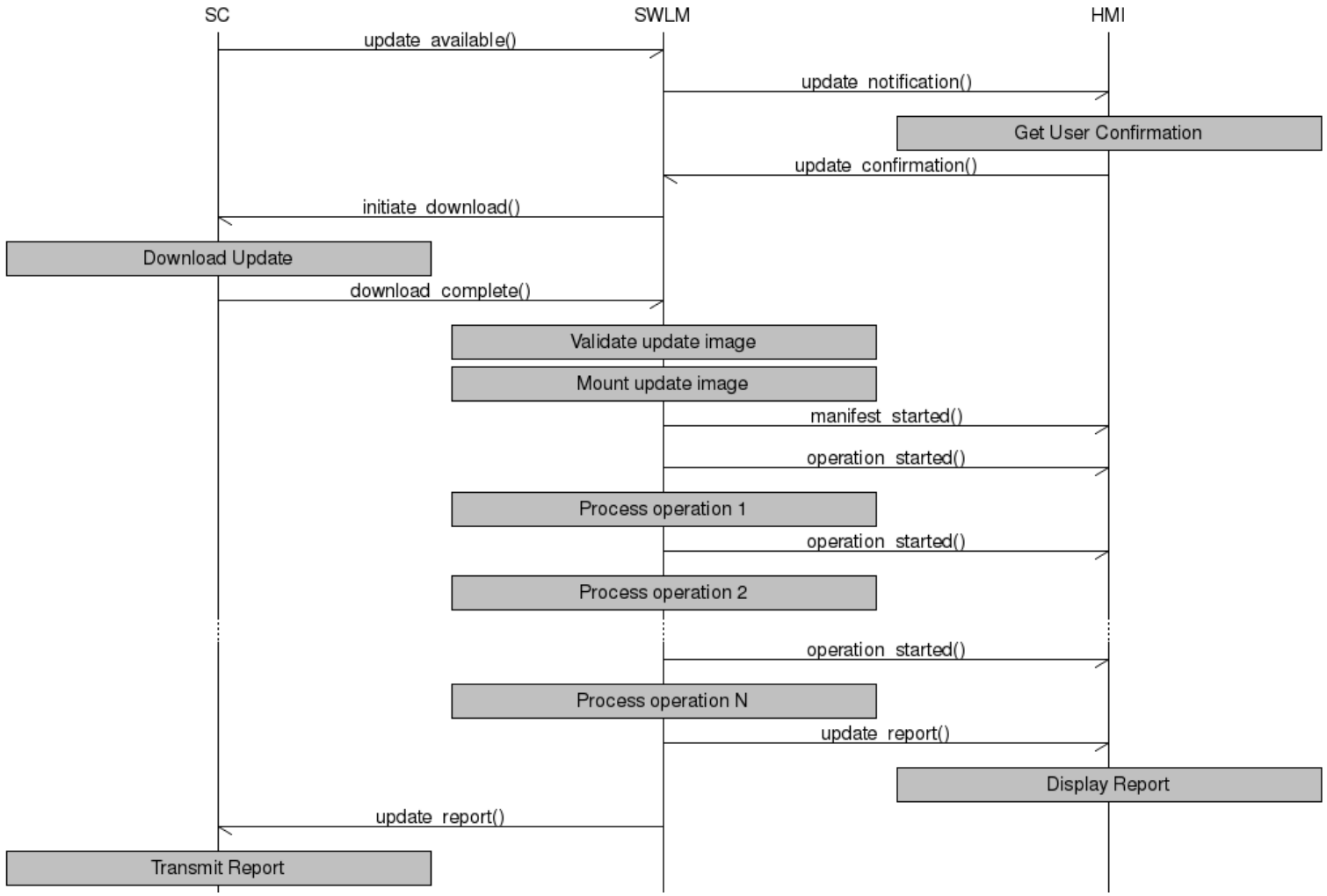
GENIVI®

Backup Slides

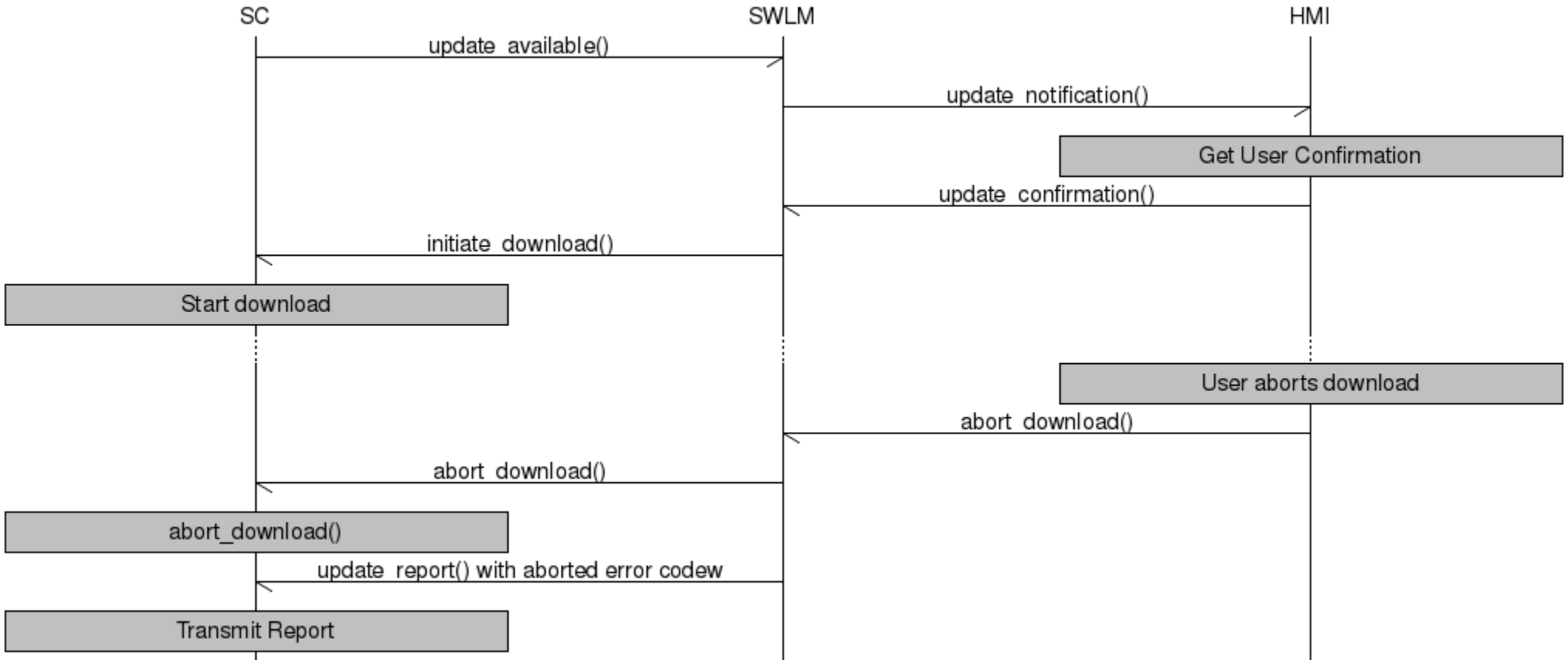
Sequence Diagrams for Use Cases

- Download Update
- Abort Download
- Update from Local Media
- Install Package
- Upgrade Package
- Remove Package
- Create Disk Partition
- Delete Disk Partition
- Resize Disk Partition
- Write Disk Partition
- Patch Disk Partition
- HMI Get Installed Software
- Sota Client Get Installed Software
- Reboot
- Start Components
- Stop Components

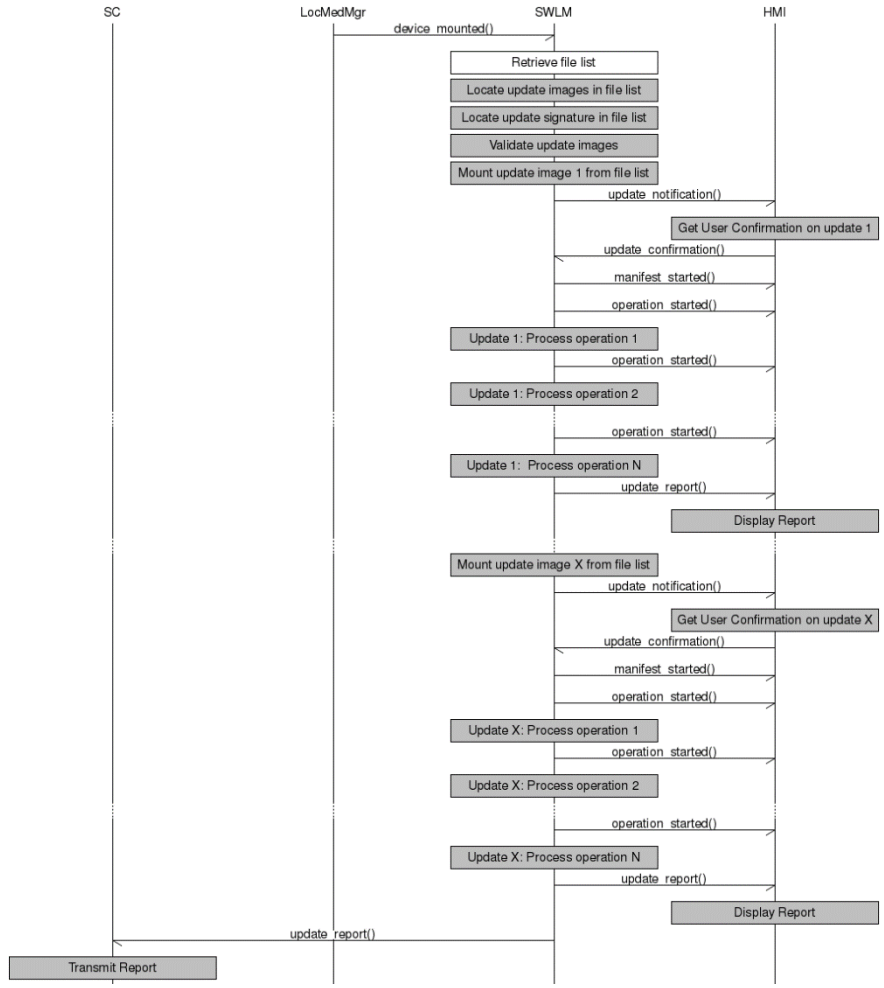
Use Case: Download Update



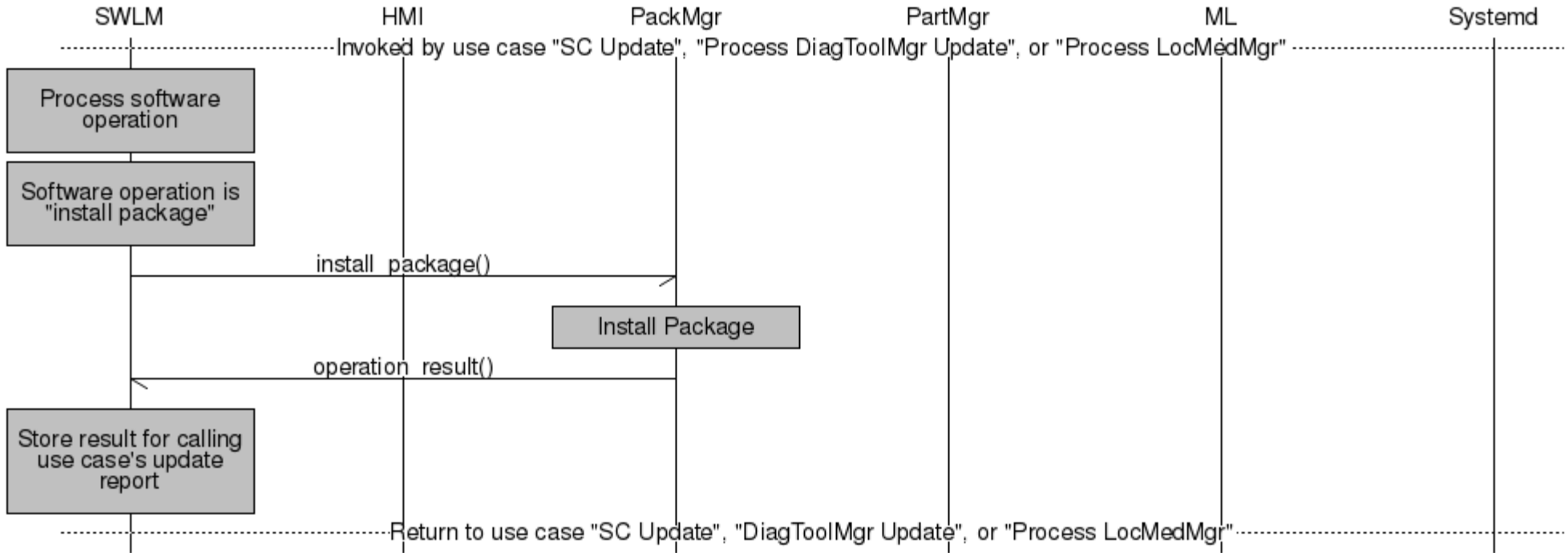
Use Case: Abort Download



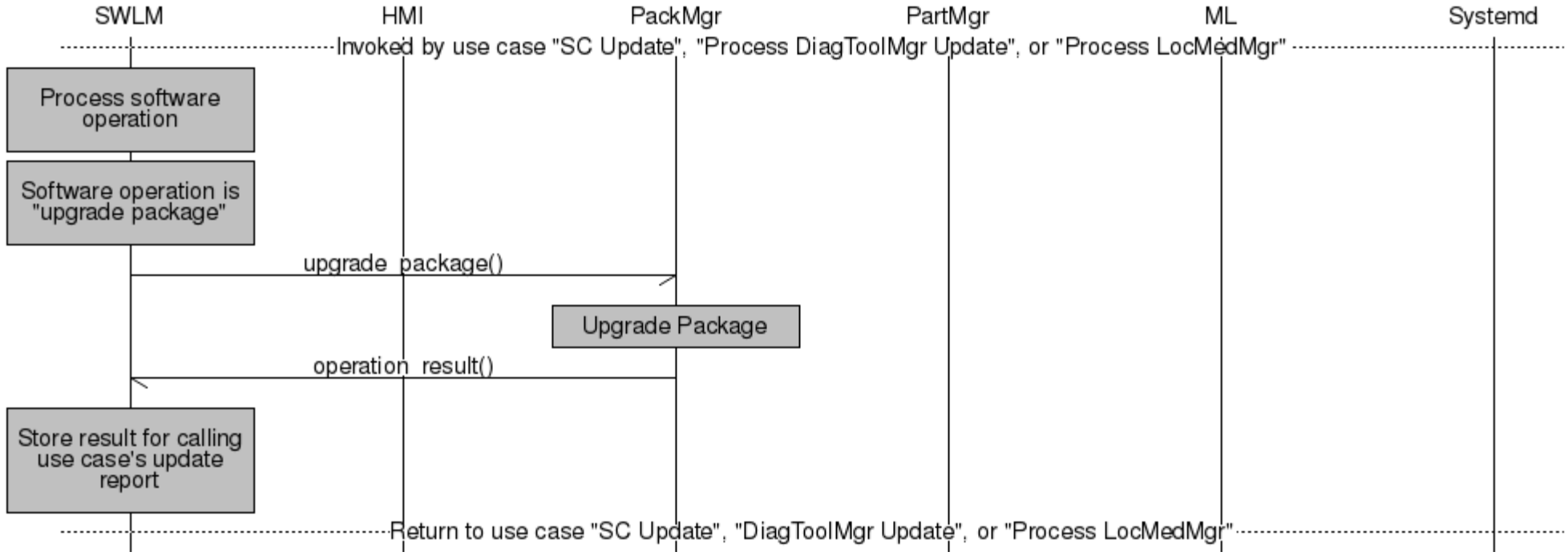
Use Case: Update from Local Media



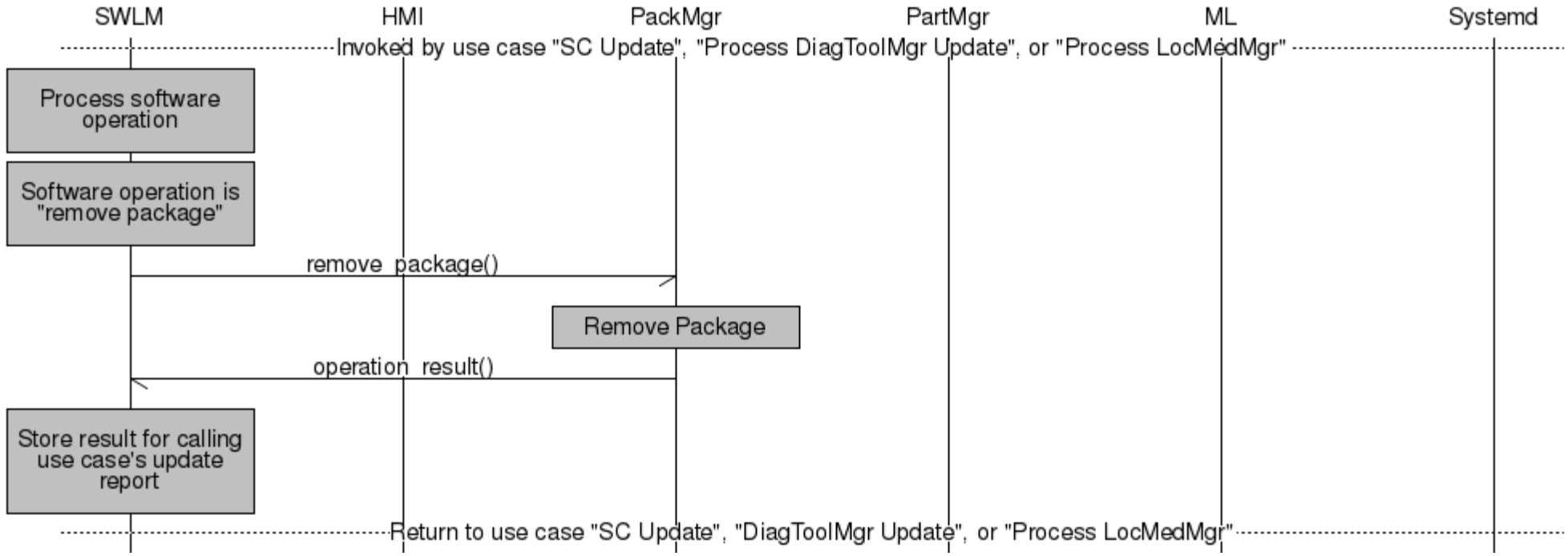
Use Case: Install Package



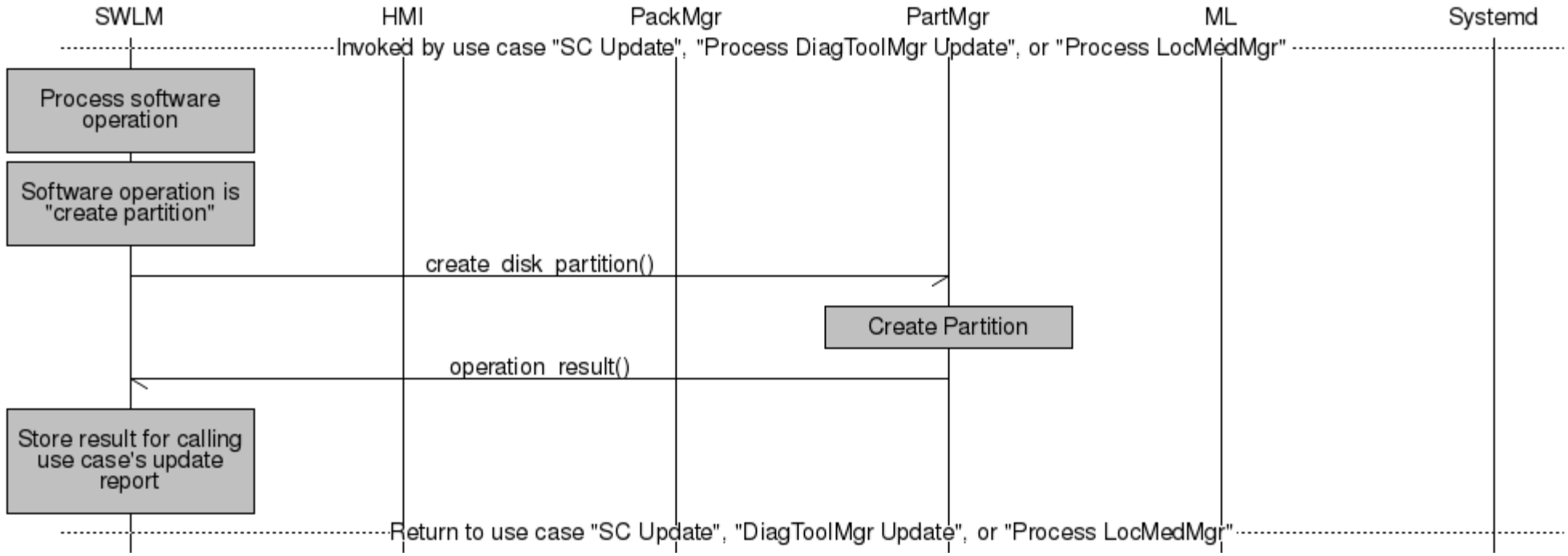
Use Case: Upgrade Package



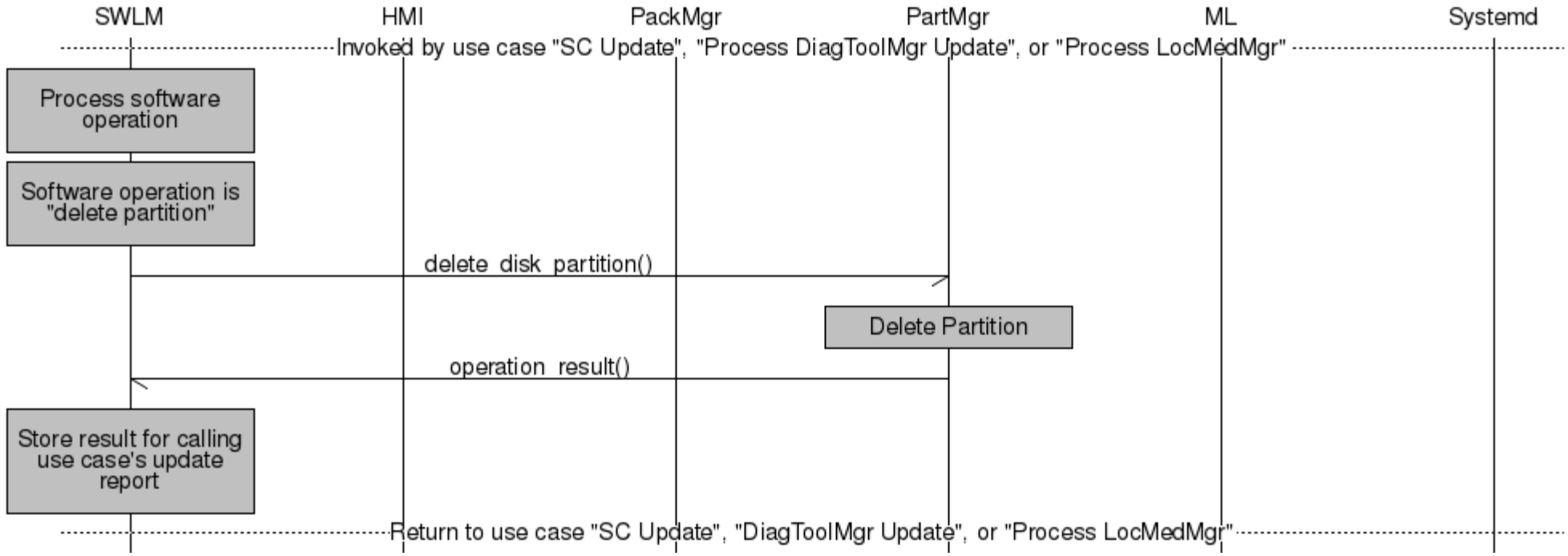
Use Case: Remove Package



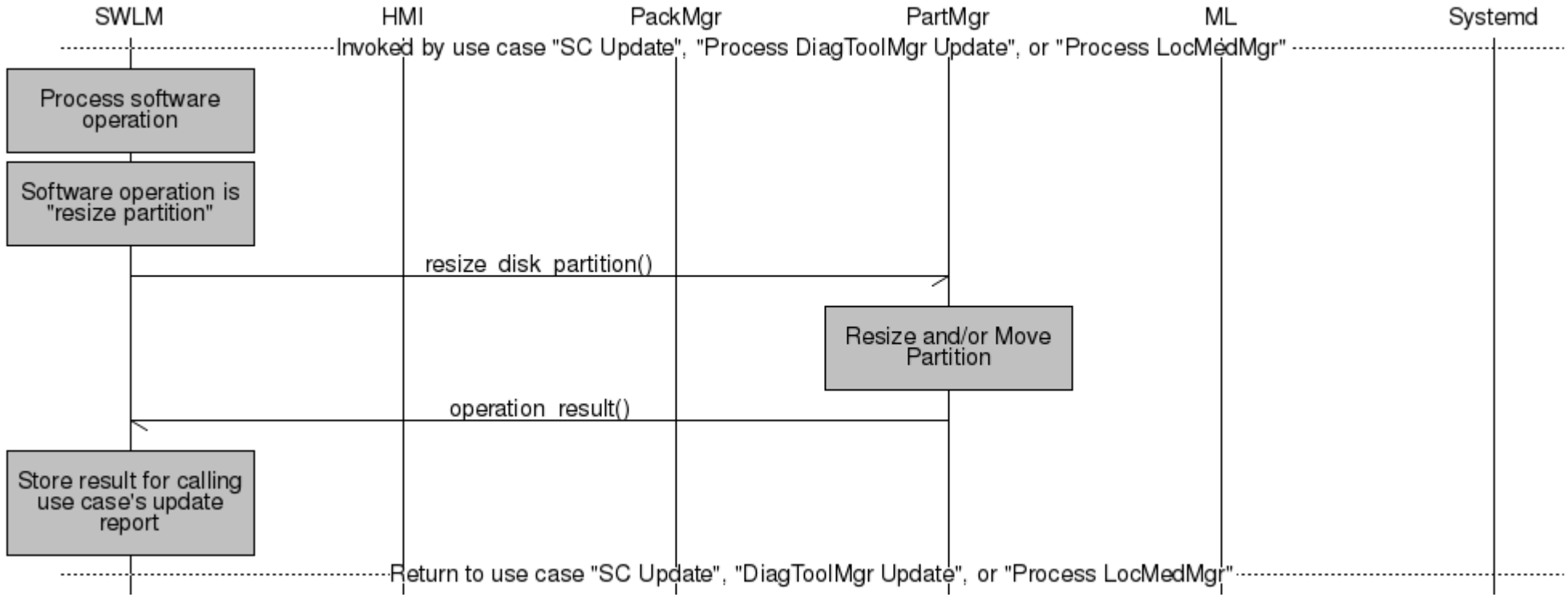
Use Case: Create Disk Partition



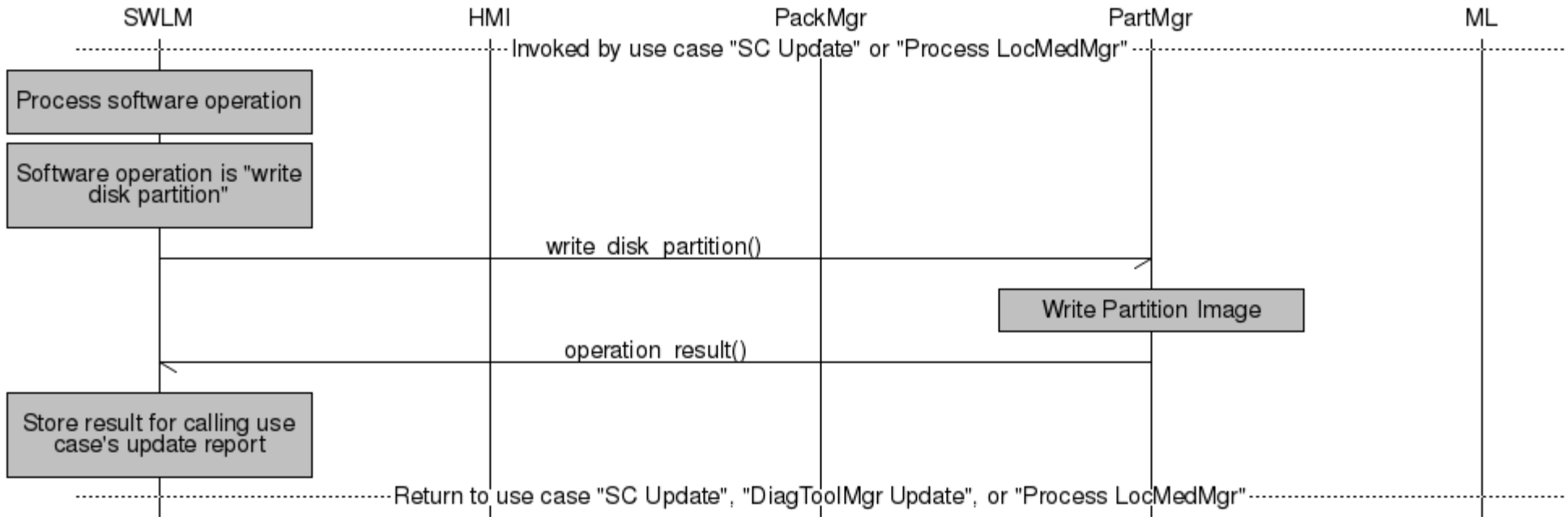
Use Case: Delete Disk Partition



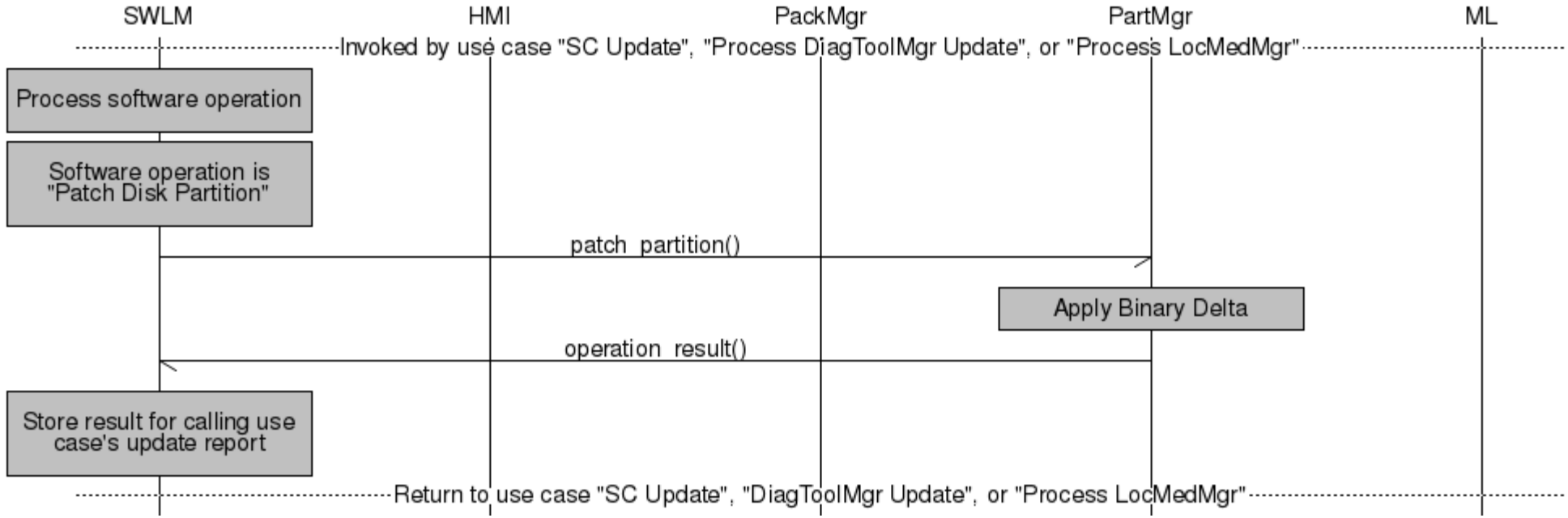
Use Case: Resize Disk Partition



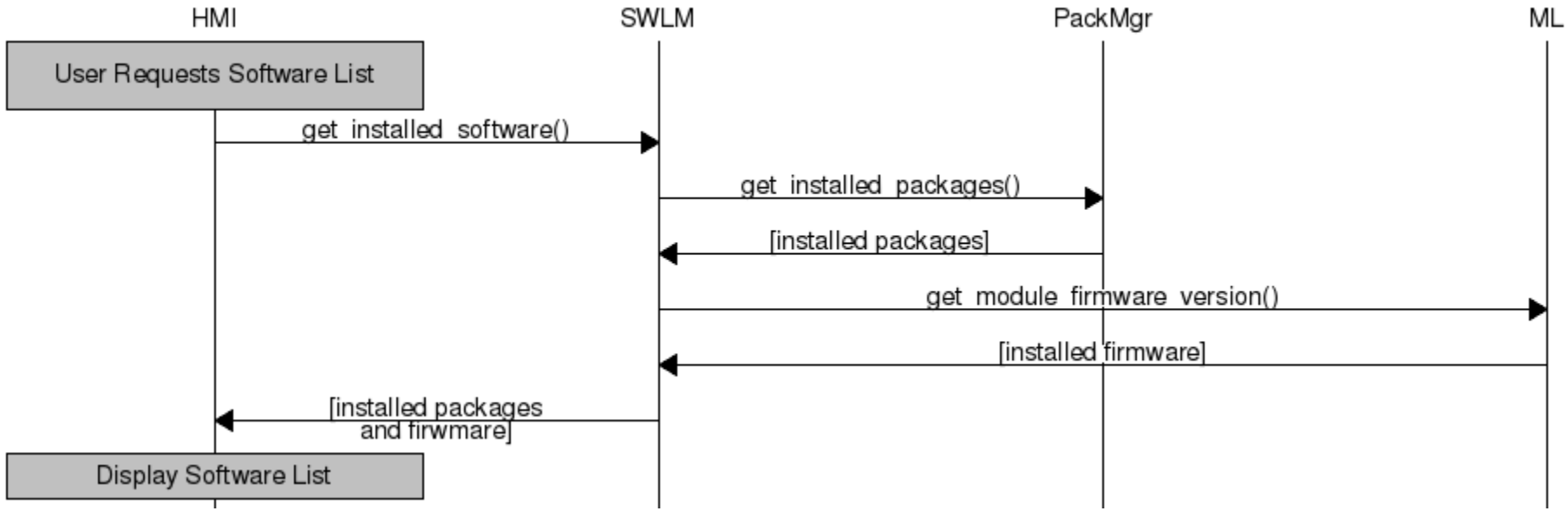
Use Case: Write Disk Partition



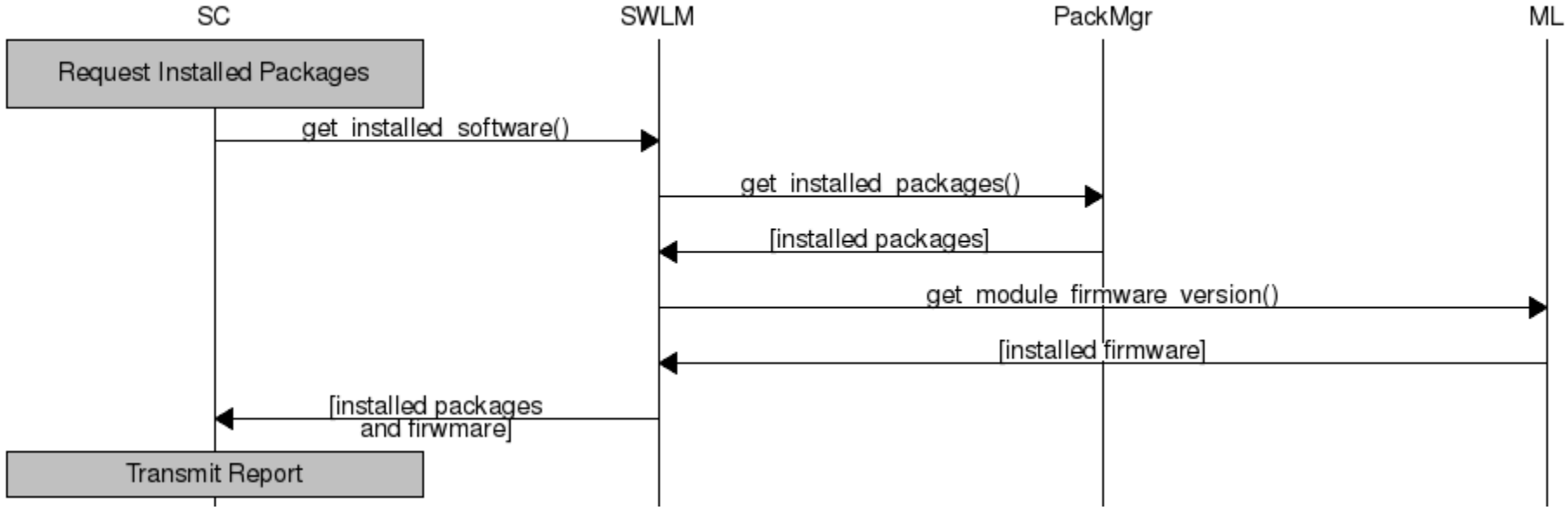
Use Case: Patch Disk Partition



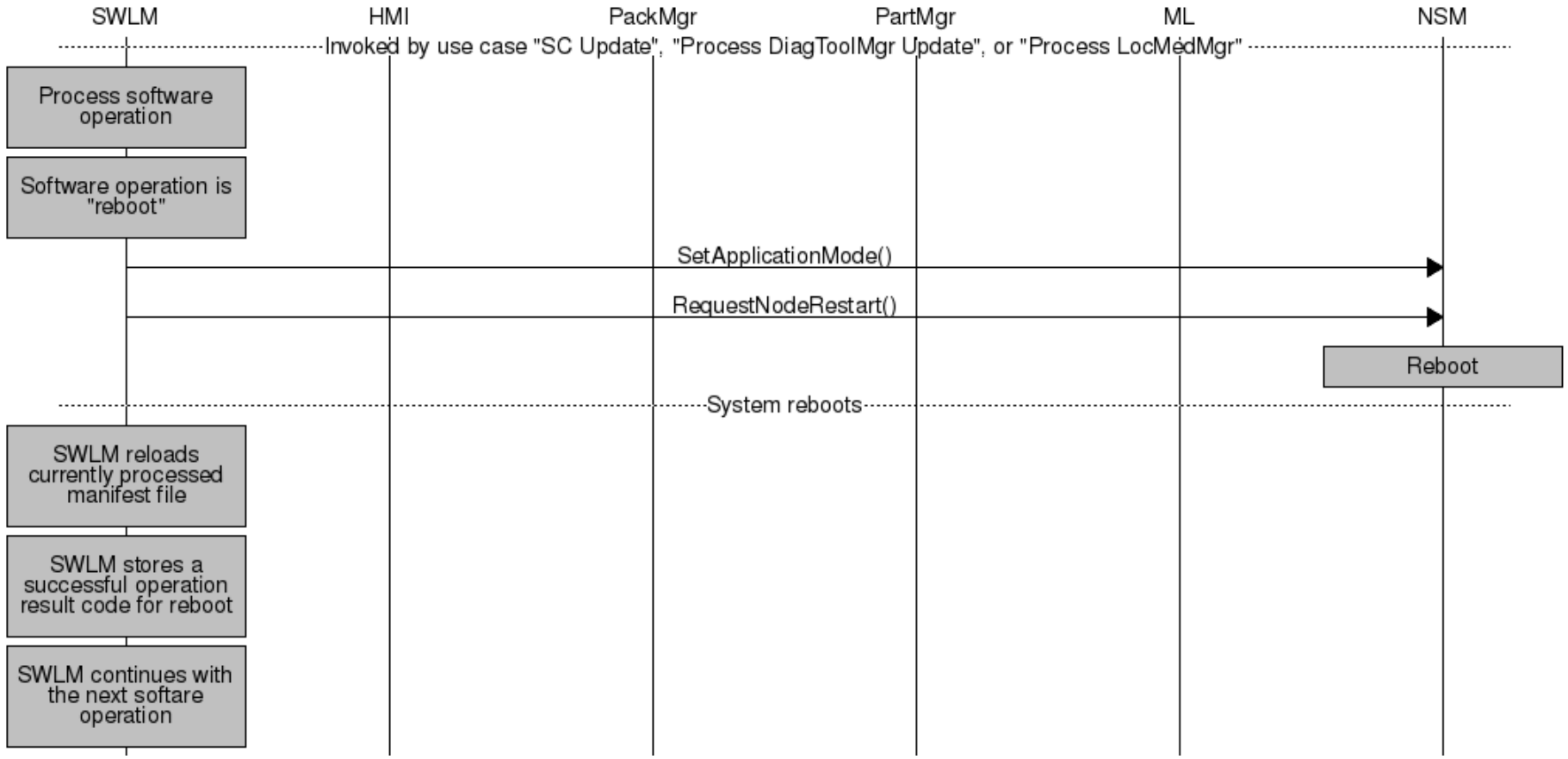
Use Case: HMI Get Installed Software



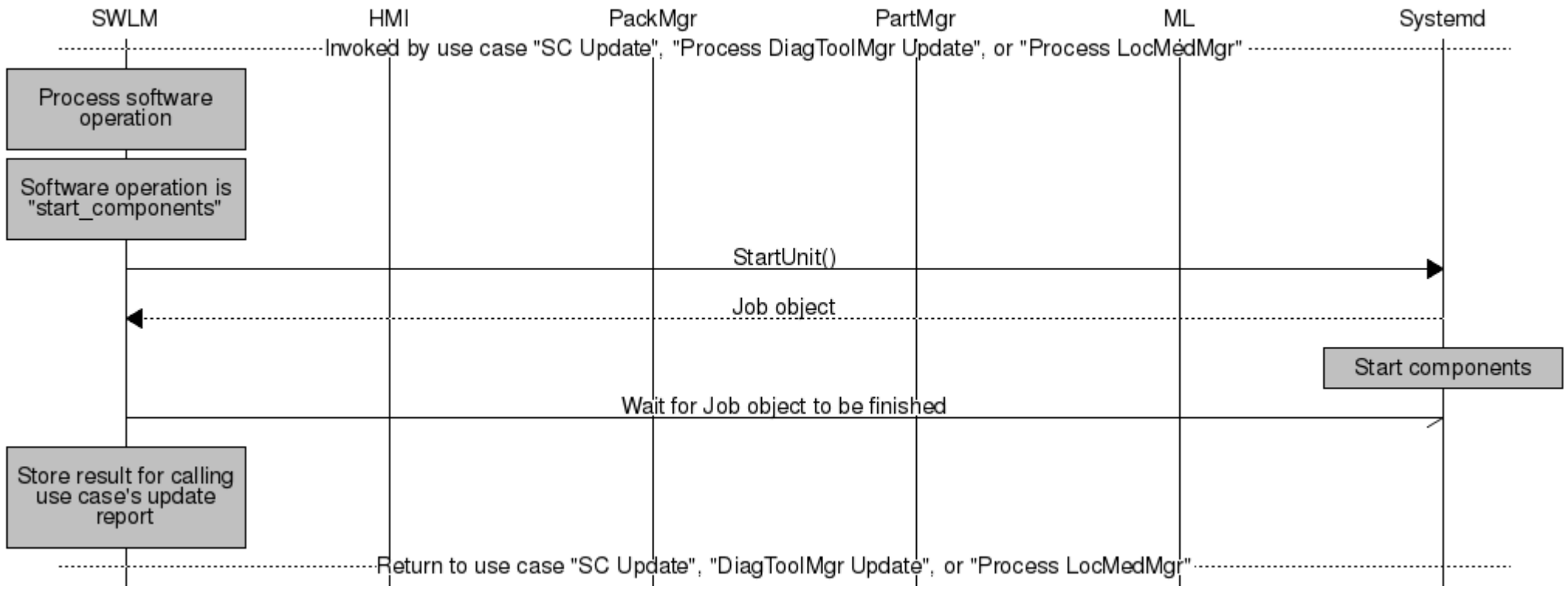
Use Case: Sota Client Get Installed Software



Use Case: Reboot



Use Case: Start Components



Use Case: Stop Components