Terms

For purposes of supporting the discussion / proposals in this document the following terms apply.

1. Asset – A set of related files tied to a string identifier. Within a single VM the Asset is tied to a specific set of responsibilities. These responsibilities may change in other VMs. As an example, CMGR exists in both Domain0 and IOVM, but does not have the same set of responsibilities on both VMs.

2. Asset Element – A subset of the files of an asset that are grouped under a specific directory. Examples of element are api, src, document, etc.

3. Asset Aspect – An instance of an asset that is tied to one or more VMs. Examples of Asset aspects: [IOVM]rmi, [Domain0]rmi, [IOVM]ipras, [Domain0]ipras, [Service]ipras, etc.

4. Product Repository – A directory structure that is created using a set of assets and their elements. The product repository consists of directories and links to files under change management.

5. Project – Top level CM Synergy project such as RAIDCore, Domain0 and Serviceability.

6. Subproject – A CM Synergy project that is included in a top level project as a member. By Convention we place the subprojects in the first directory of the project. Examples of subprojects are fwarch, Foundations_1, Platforms, ios, vios, etc.

7. Common Subproject – A CM Synergy subproject that is used across 2 or more top level projects with different names. Existing example: fwarch is used in both RAIDCore and Domain0.

8. Shared Common Subproject – A single instance of a common CM Synergy subproject that is shared between 2 or more top level projects. This is a special method of using a common subproject to optimize an individual developer's environment. It is not a special type of common subproject

9. Common Code – code that is present in 2 or more VMs. The code may or may not be used.

10. Automatic Subproject Advancement (Auto Advance) – The library build tools support a mechanism to use the latest tip version of a subproject if a newer integrated version is found during configuration. The tip version is used as the baseline for the new build. The mechanism is only on for trunk builds. If the version number of the project has a dot (.) in it, then the mechanism is disabled.

Requirements:

1. The OSA VMs will be independently buildable and versionable. (separate projects)

2. To be compatible the VMs must be built from the same version of the common code. This includes the API and any common source code. This does not specify the method used to access the common code. The code is considered to be the same version if it is identical to the code used to build the other VM.

3. Some VMs such as NAS may not use the same change management system. As such the method of sharing source and API may vary from one VM to the next. To state this explicitly, a VM that works outside of CM Synergy will not be able to use CM Synergy subprojects as a sharing mechanism and will need to import/export common code through another mechanism.

CM Synergy / Library Build Tools Behavior/Limitations

1. You cannot store a file name or a directory name with dashes in CM Synergy. We overcome this by mapping '%%' to '-' during product repository generation.

2. Changes to a common subproject do not automatically pop up in other projects that use the common subproject. Auto Advance can be used to bring these changes into library builds. When it is on, the changes checked in for one VM that uses a subproject will automatically be used in the newly created baselines of other VM projects that use the common subproject. *Note: For twigged projects the changes for common subprojects will have to be checked in against a single top level project and then cloned to the other top level projects that use the common subproject. If this is not done carefully, the subproject may branch and changes may be lost.*

3. Objects changed in one version of a common subproject will not automatically propagate to a different version of the common subproject used by another project. As an example: Domain0-wookie and RAIDCore-wrath both use the common fwarch subproject. If a file is checked out under fwarch-wookie, the change in version number will not been seen in fwarch-wrath until the RAIDCore-wrath or fwarch-wrath subproject is reconfigured to using a task that is associated with the newly checked-out object. Changes made to objects that are checked out will be seen in both projects.

4. It is possible to share a common subproject between the 2 projects. Example: Domain0-ewok and RAIDCore-boxcar both use instances of fwarch with the same baseline. The user can use the GUI to select version fwarch-boxcar in the Domain0-ewok project. At that time any new checkouts or changes in fwarch-boxcar will be visible in both projects. Care should be taken when reconfiguring either the RAIDCore or Domain0 project as it may have unintended and unexpected effects on the shared common subproject.

5. The cort tool as presently written does not cleanly support checking out /managing Domain0 projects and RAIDCore projects with the same version. Example RAIDCore-homerun is already checked out. The checkout of Domain0-homerun is noisy and not guaranteed to be correct.

6. We are not considering the use of Open reconfig to facilitate the sharing of changes across OSA VM projects under CM Synergy. It is difficult to control and produces unintended results.

7. CRs that contain changes outside a common subproject will not be automatically picked up by other projects. If the CR is targeted to the CM Synergy project for the wrong VM, the changes outside the common subproject will be lost. Example: A CR contains changes for Domain0 and the common subproject. The CR is incorrectly targeted to the RAIDCore project. The changes to the common subproject will be picked up in the next RAIDCore build and carried into the next Domain0 build. The changes to the Domain0 code will be lost and must be added to Domain0 with a clone of the original CR that is targeted to the Domain0 project. Also, The CRs that are added by Auto Advance will only be recorded in build sheets for the first project where it is incorporated.

~~6.~~

Options

Option 1 – add new assets to RAIDCore project

Pro: keeps everything in sync across assets. No compatibility matrix needed.

Con: does not allow for independent versioning of VM products.

Option 2 – new projects with no subprojects

Pro: VM products can be independently versioned. No complicated management of subproject content

Con: Maintenance problem. Any sharing of source code is done at a file level. Very easy to have missing CRs. Files can branch. It can be difficult to characterize the interface version for the compatibility matrix.

Option 3 – new projects with common subprojects

Option 3a – single common subproject per team.

Pro: VM products can be independently versioned. Sharing is at a subproject level. If two projects use the same versions of the common subprojects, they are in sync.

Con: Assets will appear in projects where they are not used. Delivery of content changes to projects can be complicated (where do I check in my changes?) It can be difficult to characterize the interface version for the compatibility matrix.

Option 3b – combination of common and uncommon subprojects per team

(current implementation)

Pro: subprojects that are only used in 1 VM are clearly identified as such.

Con: Unless we add a new sharing scope for each possible combination of asset use, there will be projects where some assets in a shared subproject are unused. If we do add new sharing scope definitions, number of sharing scopes will get unmanageable and developers will need yet another decoder ring.

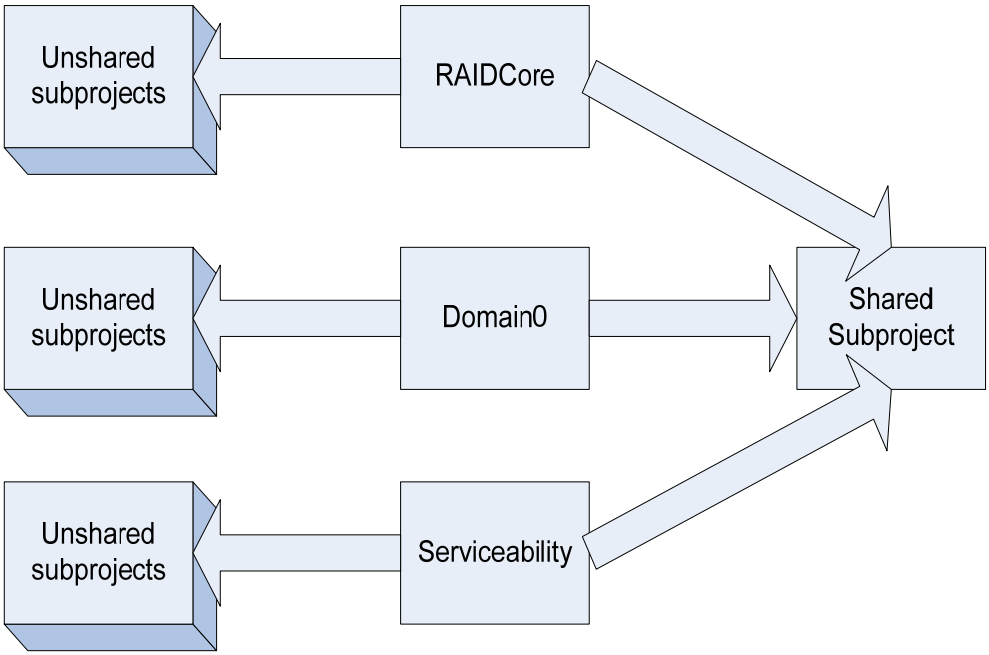
Option 3c – combination of uncommon subprojects per team and one common subproject.

(proposed new standard)

Pro: subprojects that are only used in 1 VM project are clearly identified as such. Any asset that may be common in any of the VM projects will be in the common subproject. A change to any of the common assets changes the interface version. This allows us to easily know when VMs are in sync at the interface level. We can leverage the Auto Advance features to allow the shared subproject to automatically advance to the most recent version.

Con: There will be both Linux and VxWorks implementations for some assets. These assets will be present in the project work areas. The common content for an asset team may~~will~~ be separate from the uncommon content. Auto Advance means that we cannot block changes in a VM build. Damage to one is potential damage to all. Auto Advance may confuse some developers.

This figure illustrates the proposed new scheme.



This change moves the content of the earlier named _linux and _osa sub-projects into the common OSACore subproject. Additionally, assets under ios_domain0 and cas_domain0 that are proven to be common will be moved under OSACore.

```
Domain0
  Makefile
  OSACore ← new subproject
  foundations2_domain0
  hypervisor_domain0
  platforms_domain0
```

Table 1 - New Domain0 Structure

```
RAIDCore
  OSACore ← new subproject
  foundations
  hypervisor
  platforms
etc...
```

Table 2 – Modified RAIDCore Structure

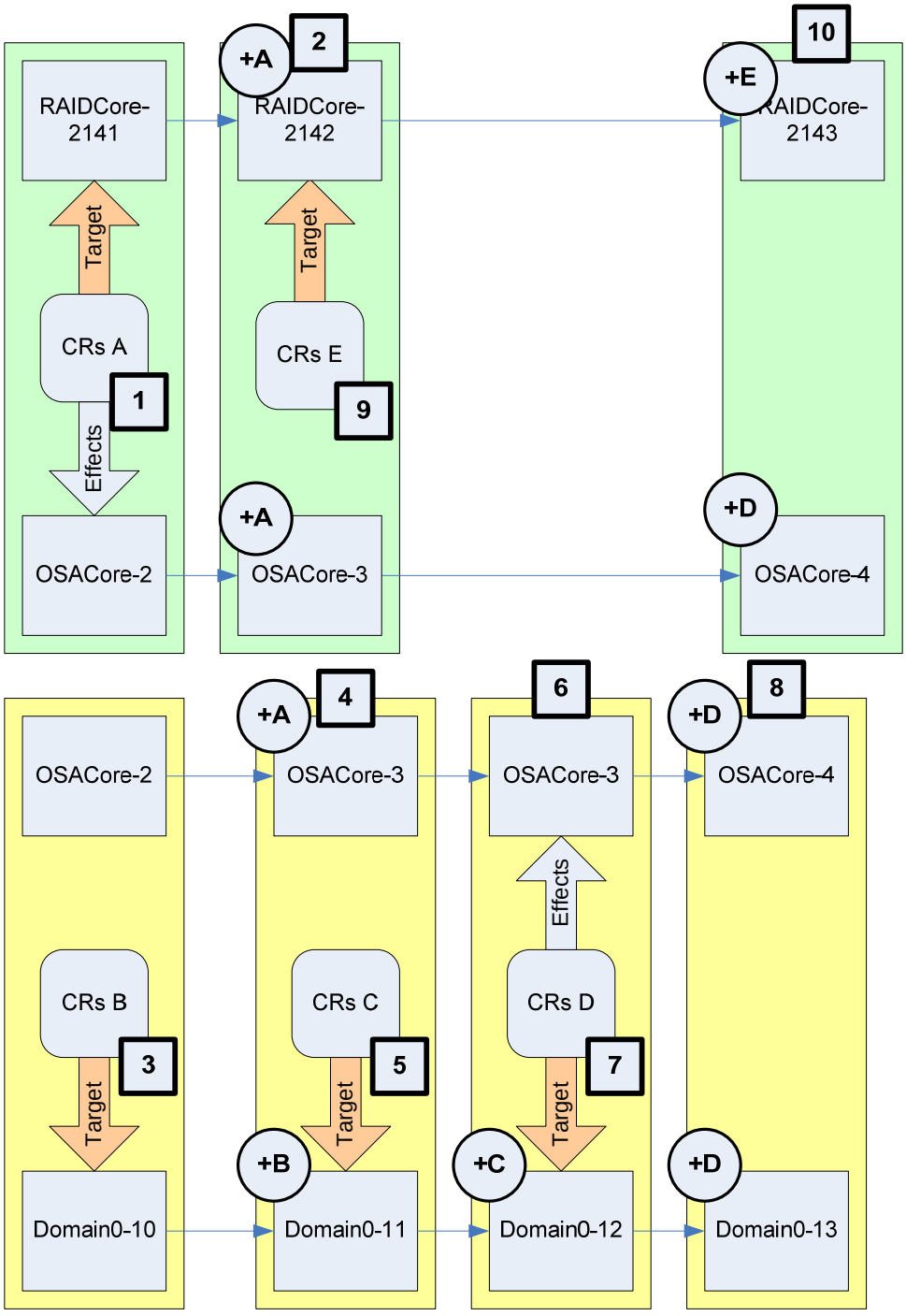
The content of OSACore is organized by team names at the top level. The below that are assets and elements. Some assets will have multiple asset aspects. For those assets, the asset file will sit under the aspect directory rather than the top level directory of the asset.

```
  cas/
    rmi/
        Common/
            rmi.asset ([Common]rmi )
            api/
        Linux
            rmi.asset ([Domain0]rmi )
            api/
            ksrc/   ← kernel space source
            usrc/   ← user space source
        VxWorks/
            rmi.asset ([IOVM]rmi )
            src/
  foundations2/
    osa/
        api/
  fwarch/
       (common assets)
  ios/
      ipras
        Linux
            ipras.asset ([Domain0,Serviceability]ipras )
            api/
            ksrc/
            usrc/
        VxWorks/
            ipras.asset ([IOVM]ipras )
            api/
            src/
```

Table 2 – Proposed OSACore structure

A possible extension of this would be to move both aspects of assets such as CMGR to the OSACore subproject.

This figure below illustrates the affects of CRs on the common subproject where automatic subproject advancement has been allowed. Specifically it shows how and when the common subprojects will advance.



In this scenario RAIDCore and Domain0 share the subproject OSACore.

1. CR group A is targeted to RAIDCore. It affects OSACore-2

2. RAIDCore-2142 is created with changes to OSACore that produce version 3. [+A indicates that both RAIDCore-2142 and OSACore-3 have accounted for CRs A.]

3. CR group B is target to Domain0. It does not affect OSACore.

4. Domain0-11 is created. It uses OSACore-3 as the baseline for OSACore. [+B indicates that Domain0-11 now accounts for CRs B.]

5. CR group C is targeted to Domain0. It does not affect OSACore.

6. Domain0-12 is created. There are no changes to OSACore. [+C indicates that Domain0-12 now accounts for CRs C.]

7. CR group D is targeted to Domain0. It affects OSACore.

8. Domain0-13 is created with changes to OSACore that produce version 4. [+D indicates that both Domain0-13 and OSACore-4 have accounted for CRs D.]

9. CR group E is targeted to RAIDCore. It does not affect OSACore.

10. RAIDCore-2143 is created and uses OSACore version 4. [+E indicates that RAIDCore-2143 now accounts for CRs E.]