DaVinci Team and Platform Support

User Manual

Sub Title

Version 2.0.0

Authors	Matthias Wernicke
Status	Released

Document Information

History

Author	Date	Version	Remarks
Matthias Wernicke	2018-03-01	1.0.2	Application note AN-ISC-8_1208
Klaus Emmert	2018-07-04	2.0.0	Document type changed to User Manual

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

1 Overview 5

2 Background 6
2.1 Collaboration (Multi-User) 6
2.2 Product Line Approach 6

3 Diff and Merge 9
3.1 Concepts 9
3.2 Project Merge 9
3.3 Merge Approaches (2-way, 3-way) 10
3.4 Object Identification 12
3.5 Project Merge in a Multi-User Development Process 13

4 Platform Functions 16
4.1 Concepts 16
4.2 Definition of Platform Functions 17
4.3 Function Assignment 17
4.4 Selective Merge based on Platform Functions 17
4.5 Tips and tricks 19

5 Library Mechanisms 20
5.1 Concepts 20
5.2 Sharing BSW Module Configurations 20
5.3 Sharing SWCs between DaVinci projects 20
5.4 Sharing SWCs from a third-party tool 22

6 Configuration Management 23
6.1 Concepts 23
6.2 DaVinci project files in CM repository 23
6.3 Working Copy 23
6.4 File Granularity 24
6.5 Branches and Synchronization Points 26

7 Glossary and Abbreviations 29
7.1 Glossary 29
7.2 Abbreviations 29

8 Contact 30

Illustrations

Figure 2-1 Collaboration 6
Figure 2-2 Product Line Approach 7
Figure 2-3 Development Process with a Platform and Several Projects 7
Figure 3-1 Project Merge Workflow 10
Figure 3-2 Two-way Merge 11
Figure 3-3 Three-way Merge 12
Figure 3-4 Development Process with Central Project Update by Integrator 14
Figure 3-5 Development Process with Partial Project Update by Domain Experts 15
Figure 4-1 Platform Functions 16
Figure 4-2 Definition of Platform Functions 17
Figure 4-3 Setup a New Customer Project 18
Figure 4-4 Derive from Platform 18
Figure 4-5 Independent Development 18
Figure 4-6 Update the Platform Functions 19
Figure 5-1 ECUC File Reference in DaVinci Configurator Pro 20
Figure 5-2 Library DCFs 21
Figure 5-3 Sharing SWCs Between a Third Party Tool and DaVinci Configurator Pro 22
Figure 6-1 Working Copy of the Repository 24
Figure 6-2 Split-File Project 25
Figure 6-3 File Status Reflected in the DaVinci Tools 25
Figure 6-4 Strategy Central Update on Trunk 27
Figure 6-5 Strategy Update Each Branch 28

1 Overview

This application note describes how to organize DaVinci projects and how to use the DaVinci tool features optimally to

- Enable collaborative development of a project with a team of developers
- Enable a product line approach

It covers aspects of

- Configuration management
- Management of libraries of AUTOSAR models (SWCs, EcuC)
- Diff and merge of AUTOSAR models
- Platform development support

Required tool versions:

- DaVinci Configurator Pro 5.16 or later
- DaVinci Developer 4.1 or later

2 Background

AUTOSAR ECU software is typically developed by a project team of several, sometimes dozens of team members. To organize the work, following approaches are typically used

- Collaboration
- Product Line Approach

These approaches also have to be applied to the DaVinci projects.

2.1 Collaboration (Multi-User)

Several project members work on the same SWC or the same BSW module. Parallel working must be possible. Reserved editing of the DaVinci project by one user is normally not accepted since it blocks the other users.

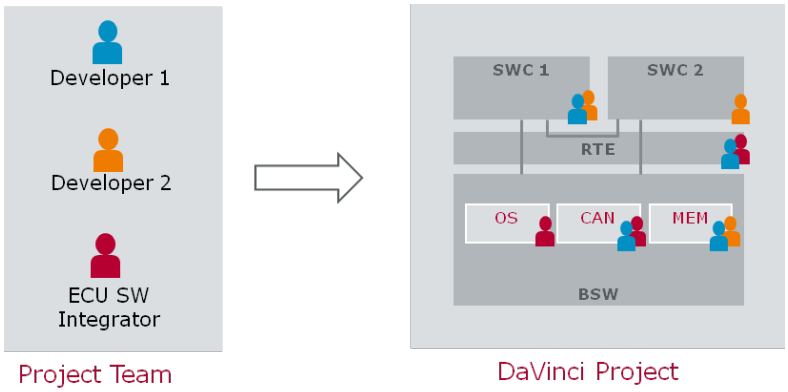


Figure 2-1 Collaboration

The DaVinci tools support collaboration with following concepts

- Diff and Merge, see section 3
- Configuration management, see section 6

2.2 Product Line Approach

A TIER1 typically offers an ECU product to several OEMs. The common parts for all OEM projects are developed within a platform project. The OEM-specific projects are derived from this platform project.

Furthermore, several ECU products may share a common library of AUTOSAR artifacts, e.g. standard data types, compu methods, etc.

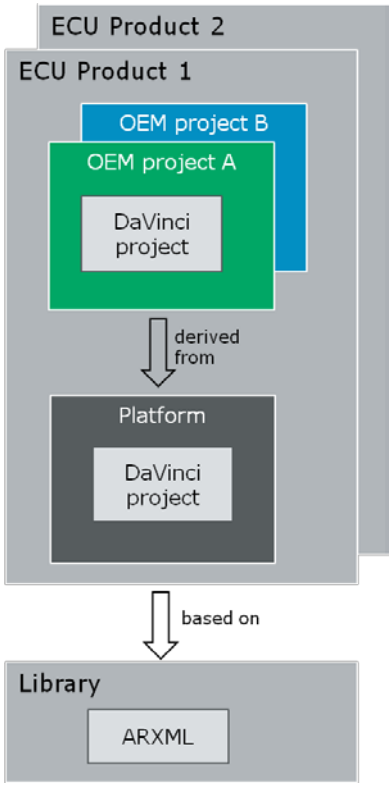


Figure 2-2 Product Line Approach

Over the time, the SW functionality of the ECU is developed as shown in Figure 2-3:

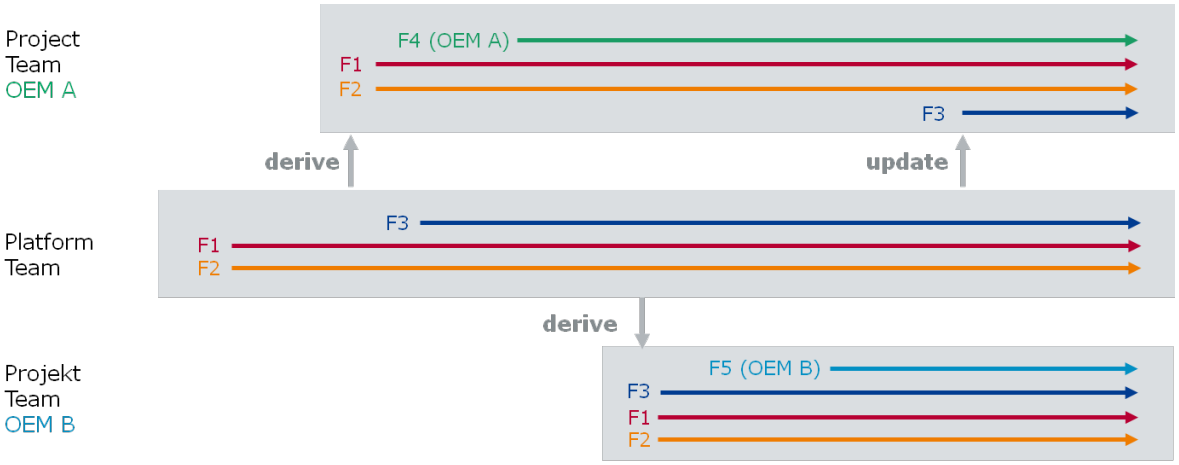


Figure 2-3 Development Process with a Platform and Several Projects

A platform team is mainly responsible for developing the central ECU functions F1, F2, F3 that make the product and are relevant for all OEMs. The goal is to provide a common set of integrated functions consisting of SWCs and matching BSW configuration.

With an independent time schedule, individual OEM project teams derive their project from the platform, and may develop OEM-specific functions F4 (OEM A) and F5 (OEM B). To benefit from the development of the platform team, the OEM projects are later on updated to the latest version of the platform. This update process shall happen under control of the

project team. It shall only affect those functions, which have been taken over from the platform. The OEM-specific functions shall not be touched.

The DaVinci tools support the product line approach with following concepts

- Platform functions, see section 4
- Library mechanisms, see section 5

Note
We assume that the BSW configuration cannot be completely derived from the SWCs, even if they are equipped with service needs. There is always an authoring step required to complete the BSW configuration. Therefore, it is not sufficient to just take over the SWCs and assume that the BSW configuration can be automatically completed in a deterministic way.

3 Diff and Merge

3.1 Concepts

A **simple** solution for team collaboration would be the attempt to place the DaVinci project folder on a shared drive, where several persons remotely access the same project folder. This will lead to race conditions and uncontrolled changes with a high risk of inconsistencies or even data corruption.

Instead, each user should have its own **private copy** of the DaVinci project. If several users modify the same AUTOSAR object in parallel in their respective working copy, a subsequent merge step is required to bring the modifications together. In contrast to e.g. C- or H-Files, a textual merge of ARXML files is not recommended. The preferred approach is a **model merge**, which is supported by the DaVinci tools.

The DaVinci tools support the diff and merge of AUTOSAR models on SWC level (with DaVinci Developer) and ECUC level (with DaVinci Configurator Pro).

Note

> In general, the diff and merge features of the DaVinci tools support split-file projects as well as single-file projects

> It is also supported to diff and merge projects that are based on different SIPs

3.2 Project Merge

The merge of a DaVinci project is controlled via an according project merge function of DaVinci Configurator Pro

> Load the **MINE** project

> Select the **OTHER** project
The AUTOSAR model inside this project will not be changed.

> Optional: Select the **BASE** project
The AUTOSAR model inside this project will not be changed.

> Launch DaVinci Developer to merge the SWCs inside the DCF workspace.
Save the DCF workspace when done. After saving, DaVinci Configurator Pro will automatically resynch.

> Continue with DaVinci Configurator Pro to merge the ECUC
Save the project when done.

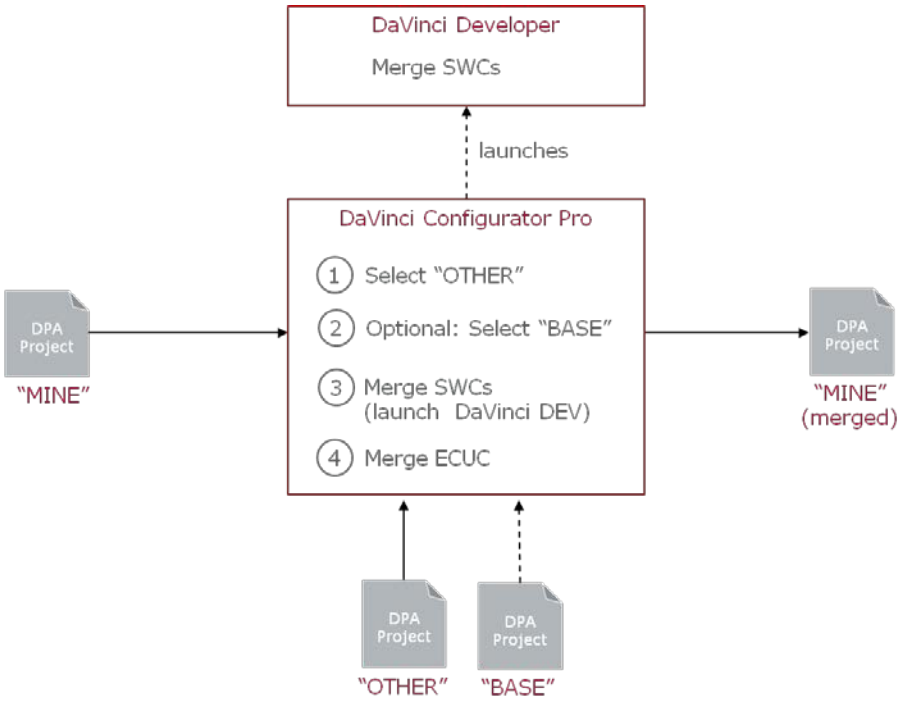


Figure 3-1 Project Merge Workflow

Note
Do not try to merge service SWCs with DaVinci Developer – like BswM, Dem etc. or other SWCs generated by DaVinci Configurator Pro. This happens indirectly by merging according module configuration with DaVinci Configurator Pro.

3.3 Merge Approaches (2-way, 3-way)

The **two-way merge** is applied when importing any ARXML files or when merging two DaVinci projects that have no common ancestor. The loaded project (**MINE**) is compared to the imported model (**OTHER**). Differences are displayed like from the perspective of **MINE**:

- Added (exists in **OTHER** but not in **MINE**)
- Removed (exists in **MINE** but not in **OTHER**)
- Modified (exists in both **MINE** and **OTHER**, but is not identical)

Merge decisions have to be met by the user.

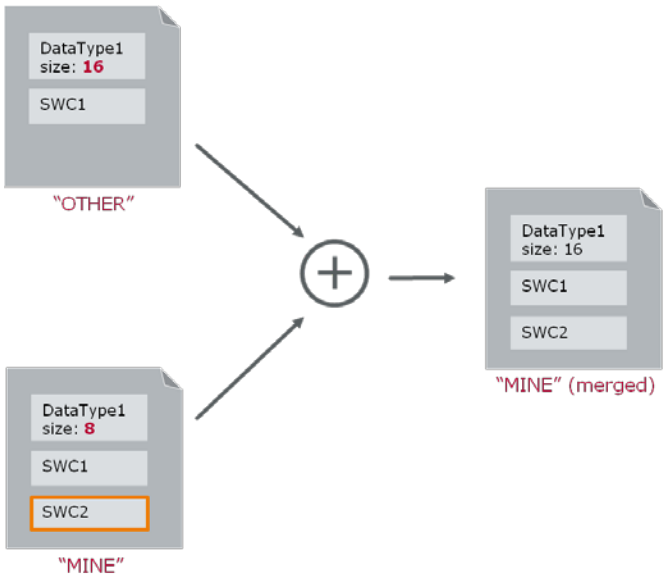


Figure 3-2 Two-way Merge

In the example in Figure 3-2 the differences are

- Modified: size of **DataType1**
- Removed: **SWC2**

The **three-way merge** is applied when merging two DaVinci projects that have a common ancestor. The loaded project (**MINE**) is compared to the imported model (**OTHER**) and the common ancestor (**BASE**). Differences are displayed like

- Removed in MINE
- Added in MINE
- Changed in MINE
- Removed in OTHER
- Added in OTHER
- Changed in OTHER
- Changed in MINE and OTHER (conflict)

Merge decisions can be made automatically. Only in case of conflict, the user has to decide.

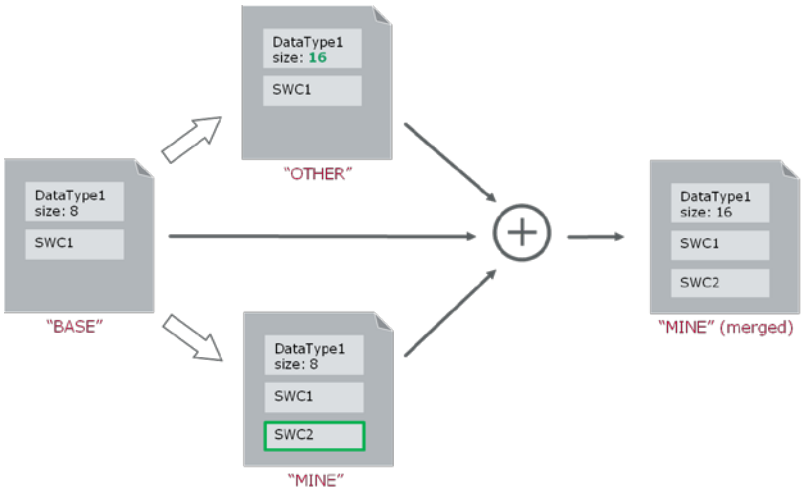


Figure 3-3 Three-way Merge

In the example in Figure 3-3 the differences are

> Changed in OTHER: size of **DataType1**

> Added in MINE: **SWC2**

Auto-merge is possible.

Limitations
DaVinci Developer 4.1 only supports the two-way merge. Three-way merge is not yet supported. DaVinci Configurator Pro supports both merge approaches. Please refer to the online help for details.

3.4 Object Identification

A special topic when merging AUTOSAR models is the identification of objects. There are two ways

> AUTOSAR path
Objects are identified based on the AUTOSAR path consisting of the parent packages and the short name. If the short name or package is changed, the object will be considered as a different object.

> UUID
Objects are identified based on their UUID independently of their short name or package.

DaVinci Configurator Pro can identify objects during project merge based on their UUID. If UUIDs are missing, it uses the AUTOSAR path instead. By using an according option, you may disable the usage of UUIDs and enforce the usage of the AUTOSAR path for object identification.

Tip
The UUID mechanism only works if the two branches to merge have been created/copied out of the same basis. Only in this case, an object will have the same UUID in both branches **MINE** and **OTHER**. If you would e.g. create a new ECUC container with same short name separately in both branches, the containers will have a different UUID. You should disable the UUID usage in this case..

Limitation
DaVinci Developer 4.1 does not yet support UUIDs during project merge. The tool identifies objects by their AUTOSAR path.

3.5 Project Merge in a Multi-User Development Process

The merge function of the DaVinci tools is not able to change those parts of the configuration, which are read-only (derived parameters, preconfigured parameters ...). Consequently, if you try to merge two projects that are based on differing input files, many differences may exist that cannot be merged. This situation can be avoided by an appropriate development process.

Let’s assume that a DaVinci project is being developed by an integrator and two domain experts (one for diagnostics, one for communication). There are two strategies:

Strategy “Central project update by integrator”

The integrator runs the project update function with the new version **V2** of the diagnostic and communication input files. Note that the project will be up-to-date with respect to the input files now, but most likely not error free.

The domain experts work now in parallel to fix the issues and complete the integration. They mainly take care about their own domain, but might need to make changes in the other domain as far as required to **get their own domain running**.

Finally, the integrator merges the projects of the domain experts one-by-one.

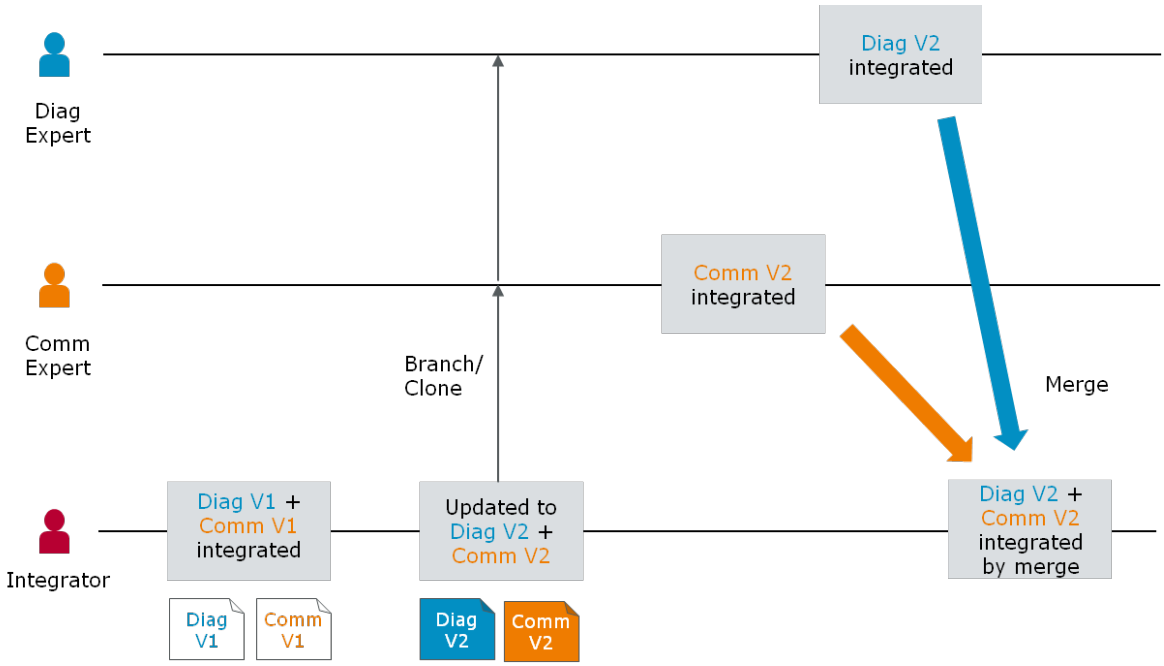


Figure 3-4 Development Process with Central Project Update by Integrator

This strategy requires only one project update step, but may increase the effort of the domain experts since they cannot fully focus on their own domain.

Strategy “Partial update by domain experts”

Basis for all is the integrated project based on a version **V1** of the diagnostic and communication input files.

The diagnostic domain expert runs the project update with the new version **V2** of the diagnostic input file and the old version **V1** of the communication input file. After that, the diagnostic expert completes the integration, focusing on the diagnostic functionality.

Vice versa, the communication domain expert runs the project update with the new version of the communication input file and the old version of the diagnostic input file. Both work in parallel.

Before merging the projects of the domain experts, the integrator runs the project update to the new version **V2** of both input files. Note that the project will be up-to-date with respect to the input files now, but most likely not error free. The integrator merges now selectively the modifications done by the domain experts. When merging the project of the communication domain expert, any differences in the diagnostic part can be ignored. Vice versa, when merging the project of the diagnostic domain expert, any differences in the communication part can be ignored.

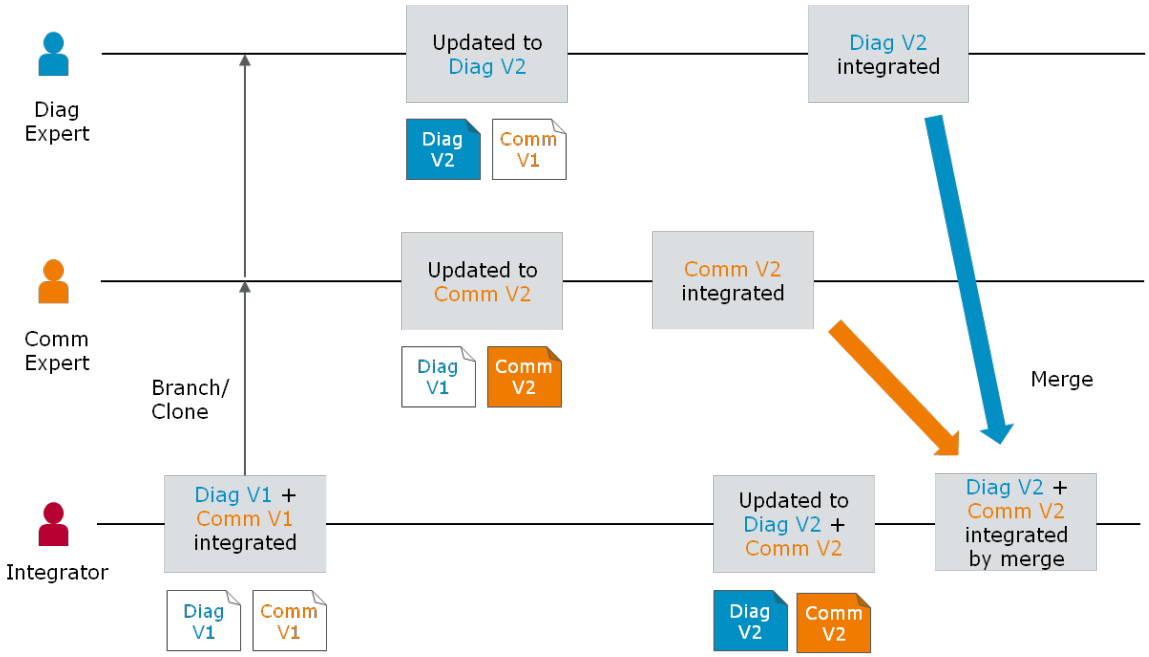


Figure 3-5 Development Process with Partial Project Update by Domain Experts

This strategy requires more project update steps, but may reduce the effort of the domain experts since they can focus on their own domain.

Please see also section 6.5 for further information how to run such development processes in combination with a CM system.

4 Platform Functions

4.1 Concepts

A platform function can be defined as a logical partition of the ECU SW, which includes parts of the SWCs and the BSW configuration.

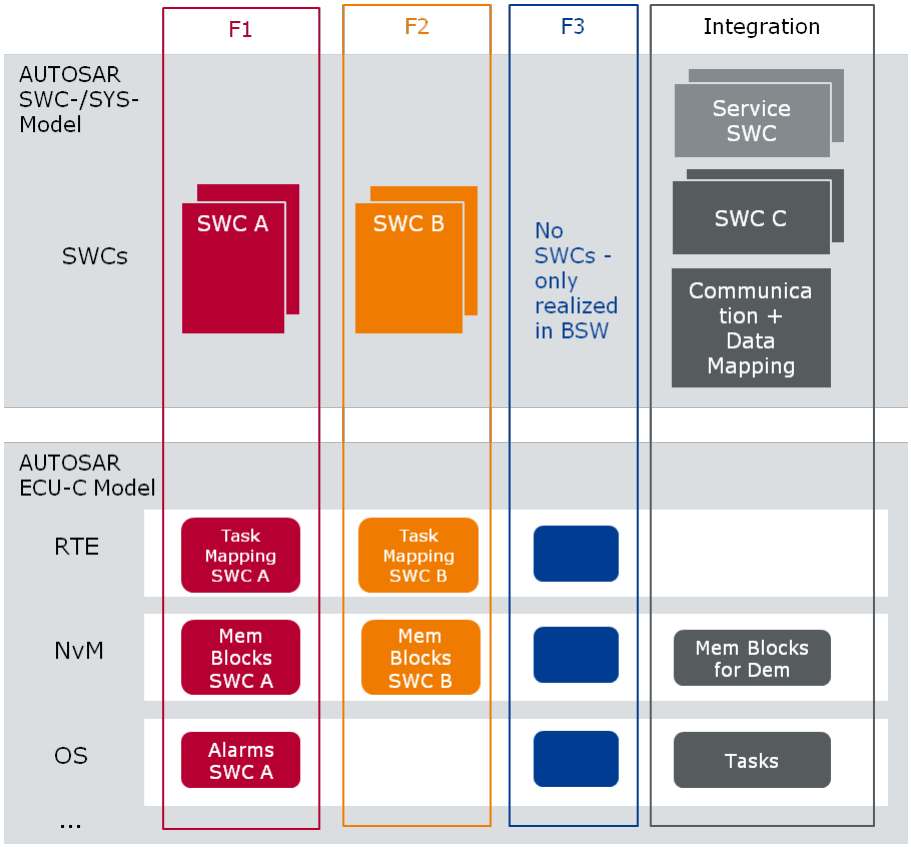


Figure 4-1 Platform Functions

In the example shown in Figure 4-1, the project consists of 3 platform functions F1, F2, F3. The platform function F1 e.g. consists of a SWC plus all related BSW configuration parts like the task mapping in the RTE configuration, the NV memory blocks in the NvM configuration and the OS alarms that finally trigger the runnable entities of the SWC. The platform function F3 is only represented by certain configuration of the BSW like standard OS tasks – no SWCs involved. **Integration** represents the project-specific part that will never be taken-over to other projects.

A platform function is mainly defined by a name, plus an additional description. Based on the name, the user is supposed to be able to identify the platform function. The user makes the initial decision about which objects belong to a particular function (explicit assignment). The DaVinci tools are able to find other related objects that would also belong to the same function (implicit assignment). This mechanism saves effort since the user does not have to make the assignment for each and every object or configuration parameter.

The logical partitioning may bring benefits during diff and merge of DaVinci projects: instead of merging the whole project in one step, the user may decide to merge the individual functions one-by-one. This makes merge decisions much easier.

4.2 Definition of Platform Functions

Platform functions can be defined with the Project Settings Editor of DaVinci Configurator Pro. They are defined with a name and description. An UUID is automatically given.

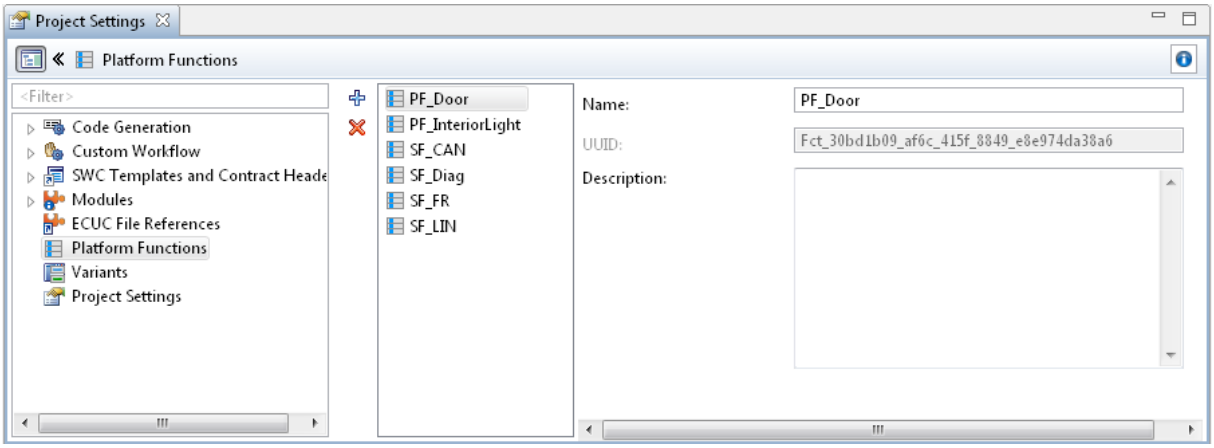


Figure 4-2 Definition of Platform Functions

4.3 Function Assignment

AUTOSAR objects on SWC-T level (e.g. SWC types, port interfaces ...) and on ECU-C level (container values, parameter values, and reference values) may be assigned to functions.

There are two kinds of function assignment:

- Explicit assignment
  An object may be explicitly assigned to one of the platform functions. The explicit assignment is done by the user. Explicit assignment to more than one platform function is not supported.
- Implicit assignment
  If an object has no explicit assignment, it may be implicitly assigned to one or more platform functions. The implicit assignment is automatically calculated by the DaVinci tools. Please refer to the Technical Reference DaVinci Platform Functions.

DaVinci Developer and DaVinci Configurator Pro provide according features to select the platform function of an object. Please refer to the online help for details.

4.4 Selective Merge based on Platform Functions

When starting the project merge with DaVinci Configurator Pro you have the option to merge either all objects, or only those objects which belong to certain platform functions.

Following sequence gives an example:

Initially, a platform project exists, partitioned into three functions F1, F2 and F3. A new customer project for OEM A is being setup. Note that the customer project may base on a different SIP than the platform project. The customer project is being updated to the OEM-specific system description.

Figure 4-3 Setup a New Customer Project

After setting-up the customer project, the platform functions F1, F2 and F3 are added. This happens via selective project merge, where only the platform functions are imported and the integration part is ignored. Note that the function definitions as well as the function assignment are automatically imported.

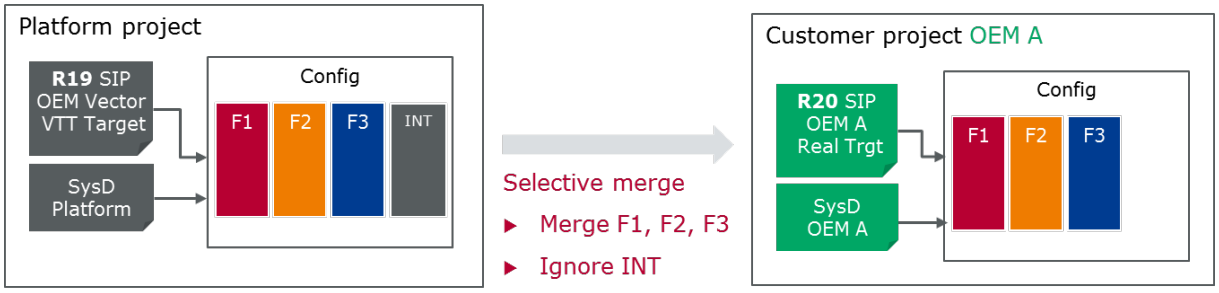


Figure 4-4 Derive from Platform

In the next phase, the projects are developed independently. The customer project is extended by completing the integration part. Also, F3 had to be changed inside the customer project. Meanwhile, the platform functions F1, F2 and F3 are extended inside the platform project.

Figure 4-5 Independent Development

Finally, the customer project is updated to the new version of the platform functions. Again, this happens via a selective project merge. It may happen either in one step by selecting all three platform functions, or step-by-step by first merging F1, then F2, then F3. Knowing that F1 and F2 have not been changed inside the customer project, the merge decision is easy: take-over all modifications. Only for F3 an individual review/merge decision is required.

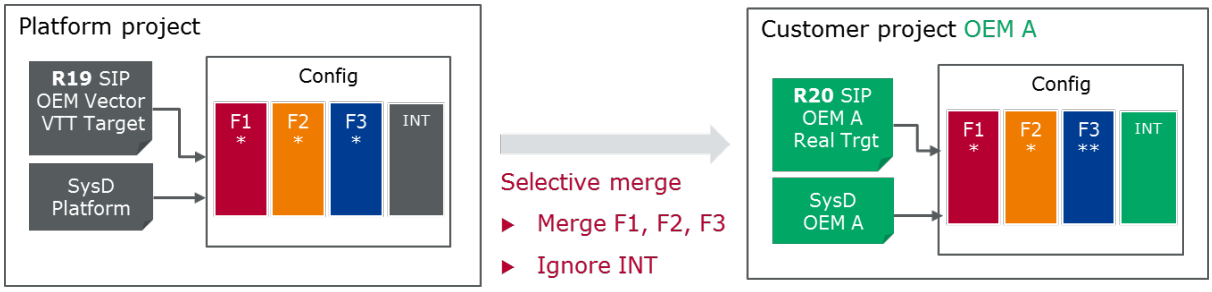


Figure 4-6 Update the Platform Functions

The selective merge approach brings following benefits

- Less differences
  Since only a subset of the data is being diffed, the number of differences is smaller and easier to handle
- Easier merge decisions
  If only individual platform functions are selected for merge, it may be easier to make merge decisions as mass operation

4.5 Tips and tricks

Here are some tips about using the platform function concept

- Number of platform functions
  The number of function definitions is not limited. However, a balance must be found between the effort for making the function assignment and the expected saved effort during project merge. A reasonable number of platform functions may be 2-20. In some cases, it may be even helpful to just have one single function **Design** to mark those parts of a project, which should be reused.
- Do not create the same function definition in several projects manually
  The function definitions are automatically imported. You don’t need to define/copy them manually in several projects. This will even lead to problems: each function definition you create will get an own UUID, which is used for identification. The DaVinci tools will warn you if you try to merge two projects, where a function definition has the same name but different UUID.
- Use case: take over a SWC including their task mapping
  This can be achieved by explicitly assigning the SWC type to a function via DaVinci Developer. The associated **Rte/RteSwComponentInstance** containers, which contain the task mapping, will be implicitly assigned.
- Use case: take over parts of the BswM configuration
  This can be achieved by explicitly assigning the mode rules via the BSW Management Editor of DaVinci Configurator Pro. The related mode request ports, expressions, actions etc. will be implicitly assigned.

5 Library Mechanisms

5.1 Concepts

A library in terms of the DaVinci tools is a set of ARXML files, which contribute read-only model parts to the DaVinci project. By using according tool features of DaVinci Configurator Pro and DaVinci Developer you are guaranteed that the content is displayed read-only and that the files remain unchanged when saving the project.

5.2 Sharing BSW Module Configurations

The ECUC File Reference mechanism of DaVinci Configurator Pro enables the definition of module configurations that can be shared by other projects. The master project, which has to be configured as split-file project, is used for editing this module configuration. A slave project references individual ECUC split files of the master project. The content of these files are loaded with the project and displayed read-only. As a result, the module configurations in these files appear in the slave project and are part of the validation and generation process like any other **local** module configuration.

In the example in Figure 5-1, the MCU module configuration has to be identical in Project1 and Project2. Project1 is used for editing the MCU configuration, so it has the role of the master project. Project2 references the split-file with the MCU configuration.

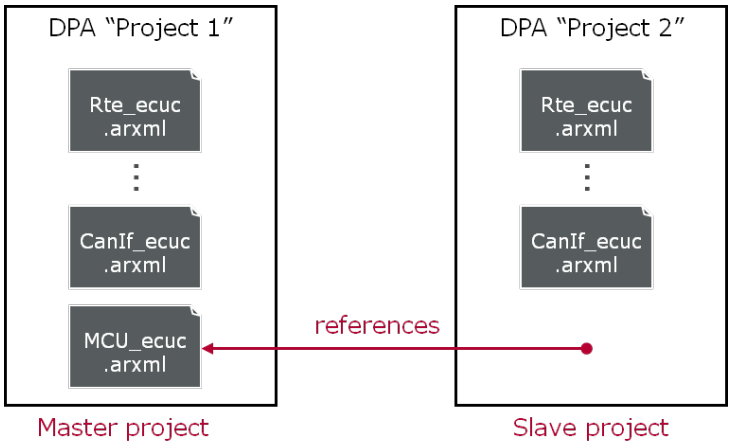


Figure 5-1 ECUC File Reference in DaVinci Configurator Pro

5.3 Sharing SWCs between DaVinci projects

A DCF workspace may be built-up hierarchically by including other DCF workspaces. DaVinci Developer recursively loads the content of all included DCFs. This content will always be display read-only.

To support a product line approach, the DCFs could be e.g. organized as shown in Figure 5-2:

> A DCF workspace contains those design elements, which are very stable and shared by several product lines (company standards). This may include a harmonized set of data types or compu methods.

> For each product line, a specific DCF workspace exists that includes the company standards DCF workspace. The product line DCF contains e.g. SWCs that are common for all projects belonging to that product line.

> For each project, a specific DCF workspace exists that includes one of the product line DCF workspaces. The project DCF contains e.g. the ECU composition SWC and other integration-specific SWCs.
Only the project DCF is directly part of a DPA project.

Depending on the features of the CM tool, the hierarchical DCF structure could also be represented by an according folder structure, or as packages/components in terms of the CM system.

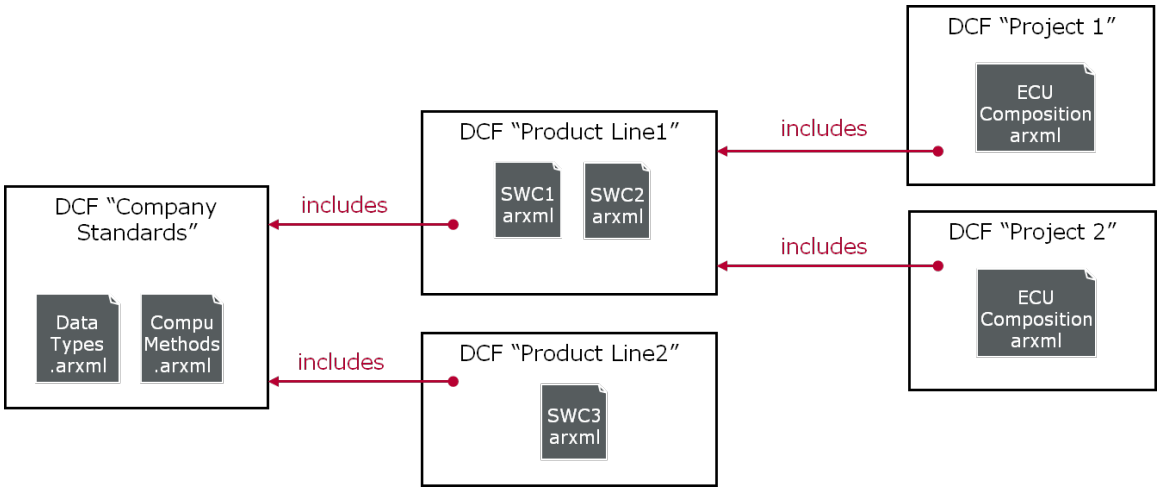


Figure 5-2 Library DCFs

Note
SWCs that are imported from input files (i.e. created/updated by DaVinci Configurator Pro during project update) should remain in the project DCF. You should not move them to a Library-DCF.

Reason: Since the content of a sub-DCF is read-only from perspective of the project DCF, the update of such SWCs would fail.

Hints for diff/merge
When using the project merge function for a DCF that includes library DCFs, the diff results will also cover diffs in the library DCFs. Please be aware that the content of a library DCF is read-only, so you will not be able to apply merge decisions.

The diff/merge of a library DCF can be done as separate step. You may open the library DCF directly with DaVinci Developer and use DaVinci Developer's DCF merge function.

Limitations for usage of platform function concept
The usage of platform functions is only possible within the project DCF. The library DCFs are not directly linked to a DPA project and therefore have no access to the platform function definitions.

This limitation may be overcome with a future version of DaVinci Developer.

5.4 Sharing SWCs from a third-party tool

DaVinci Configurator Pro can be used in combination with a third-party SWC tool with following approach:

> The Application Components folder contains the SWC files created by the third-party SWC tool. DaVinci Configurator Pro loads the content of these files read-only.

> The Service Components folder contains the SWCs created by DaVinci Configurator Pro (indirectly created by the module code generators). The third party tool may load the content of this folder but must not change the files.

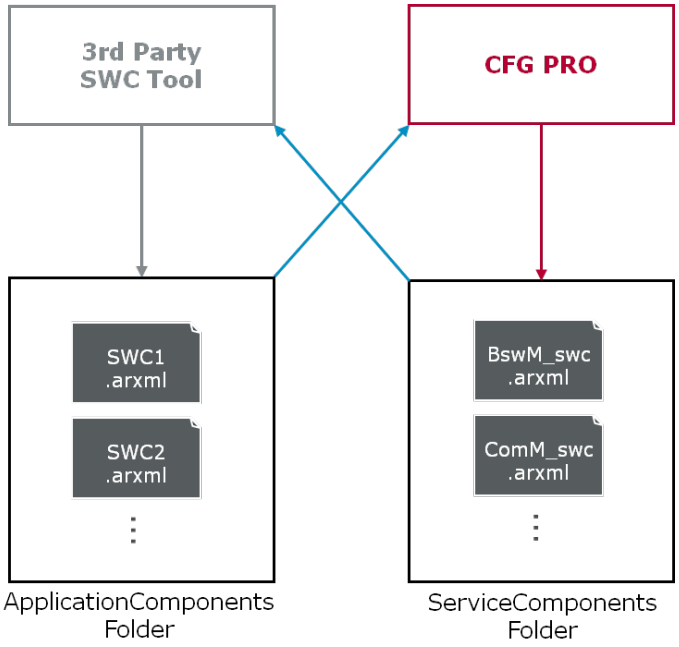


Figure 5-3 Sharing SWCs Between a Third Party Tool and DaVinci Configurator Pro

6 Configuration Management

6.1 Concepts

As soon as several team members are involved or several projects have to be handled, the usage of a CM system is recommended. The DaVinci tools provide no own CM capabilities. Instead, they cooperate with third party CM tools based on a file interface. The CM operations are managed outside of the DaVinci tools via according features of the CM tool.

It is essential to use the right strategy of branches and synchronization points to reduce the merge effort.

6.2 DaVinci project files in CM repository

When committing a DaVinci project to the CM repository, following files may be excluded because they are generated:

- Module files folder (default: **.\Appl\GenData**)
- VTT module files folder (default: **.\Appl\GenDataVip**)
- Backup files (**.\Backups**)
- Measurement and Calibration Files folder (default: **.\Config\McData**)
  Note: Only the Master.a2l file, which is authored by the user, should be stored on CM
- BSW Internal Behavior Files (default **.\Config\InternalBehavior**)

Note
If you store those files anyway in CM, please be aware that they may be changed during code generation should not be write-protected inside the working copy.

Furthermore, the file **<Project name>.<User name>.silent.dcuser** parallel to the `.dpa` file needs not to be committed. It contains user specific preferences like window sizes etc.

6.3 Working Copy

The DaVinci tools have no direct interface to the CM repository. Instead, each user should work on a private working copy of the DaVinci project. The DaVinci tools load/save the files from this working copy.

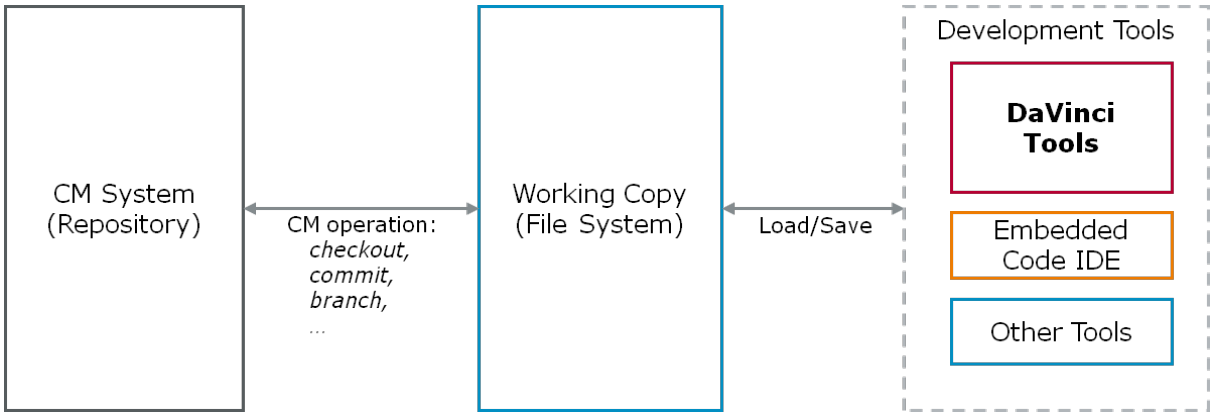


Figure 6-1 Working Copy of the Repository

For performance reason it is recommended to have the working copy in the local file system.

Note
When working with several branches or revisions of the same project (see also section 6.5), a separate working copy for each branch should be used. This allows you to e.g. open one branch of the project with the DaVinci tools, and select other branches or revisions for diff/merge.

6.4 File Granularity

The DaVinci tools support the AUTOSAR split-file concept, where the overall AUTOSAR model is distributed over several ARXML files.

> DaVinci Developer supports the storage of a workspace in several split-files, e.g. one file per SWC type.

> DaVinci Configurator Pro supports the storage of an ECU configuration either as monolithic file, or split into one file per BSW module.

Using the CM tool's features, you may control the access mode of the individual files (read-only vs. read-write). Following central files are most likely changed by all users. So, they have to be r/w to enable changes:

> FlatExtract and FlatMap in the System folder

> InitialEcuC in the ECUC folder

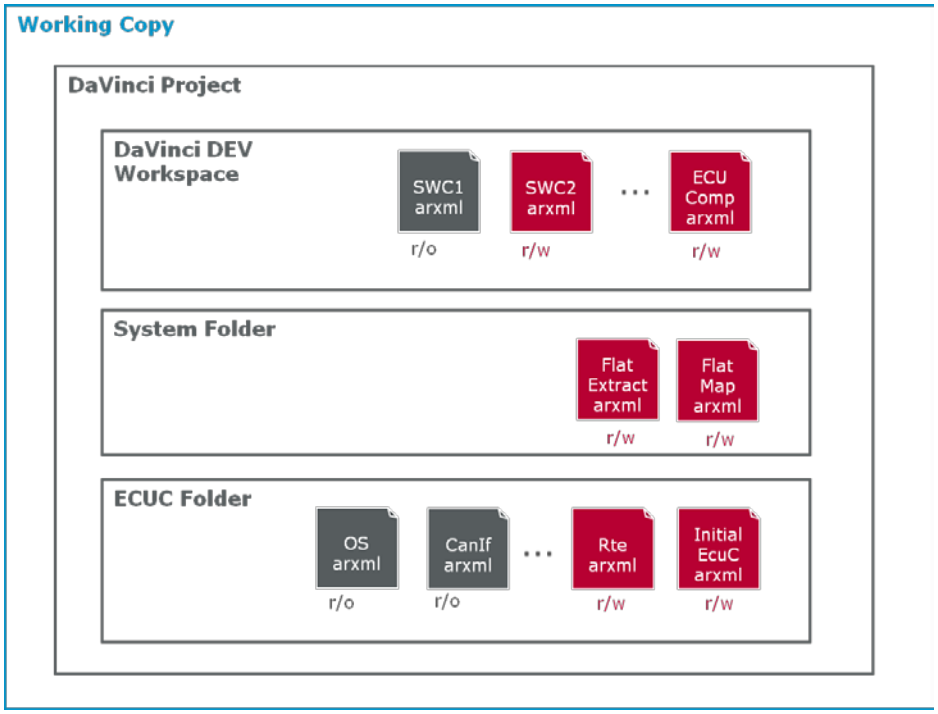
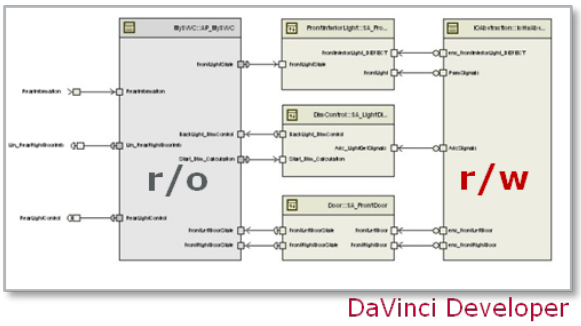


Figure 6-2 Split-File Project

It is recommended to keep the file granularity stable over the lifetime of the project. This avoids rework of the CM repository structure.

The file status of the ARXML files in the DaVinci project is reflected in the GUI of the DaVinci tools.



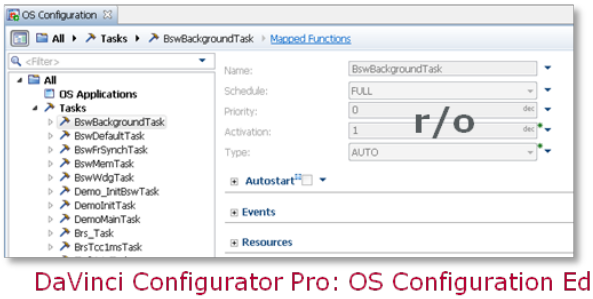


Figure 6-3 File Status Reflected in the DaVinci Tools

Note
Project merge works with any file granularity. Project merge is supported with monolithic files as well as split files.

6.5 Branches and Synchronization Points

We use following terms

Sync-Merge

The sync-merge is a process where a branch is being prepared for being integrated into the trunk. The sync-merge consists of three parts:

- Merge of the AUTOSAR model performed by the DaVinci tools using the three-way-merge approach, see section 3.
- Registration of the merge by means of the CM system, without performing an actual merge on file level (this is done by the DaVinci tool instead).
  In SVN: Use **merge to head revision** with option **only record the merge**
- Commit the modified files to the branch

Commit-Merge

The commit-merge is a process where a branch is integrated into the trunk. The commit-merge consists of two parts:

- Registration of the merge by means of the CM system, without performing an actual merge on file level.
- Commit the files to the trunk

Following diagrams show the possible branch/merge strategies. Legend:

- Light grey dot: version/revision of the project without change of input files
- Dark grey dot: version/revision of the project including change of input files (project update)
- Dot with **S**: sync-merged version/revision of the project
- Dot with **C**: commit-merged version/revision of the project

Strategy “Central Update on Trunk”:

Project updates are only performed in the trunk. After the project update, a branch for each user is created. Each user works in his branch and may create revisions. After finishing the development of the user branches, the branches are merged one-by-one: sync-merge, then commit-merge, then terminate the branch. After merging all branches, the next project

update may be executed on the trunk. After the project update, new user branches are created.

This strategy reduces the number of project updates, but blocks the users while their branches are “dead” during merge and project update on the trunk.

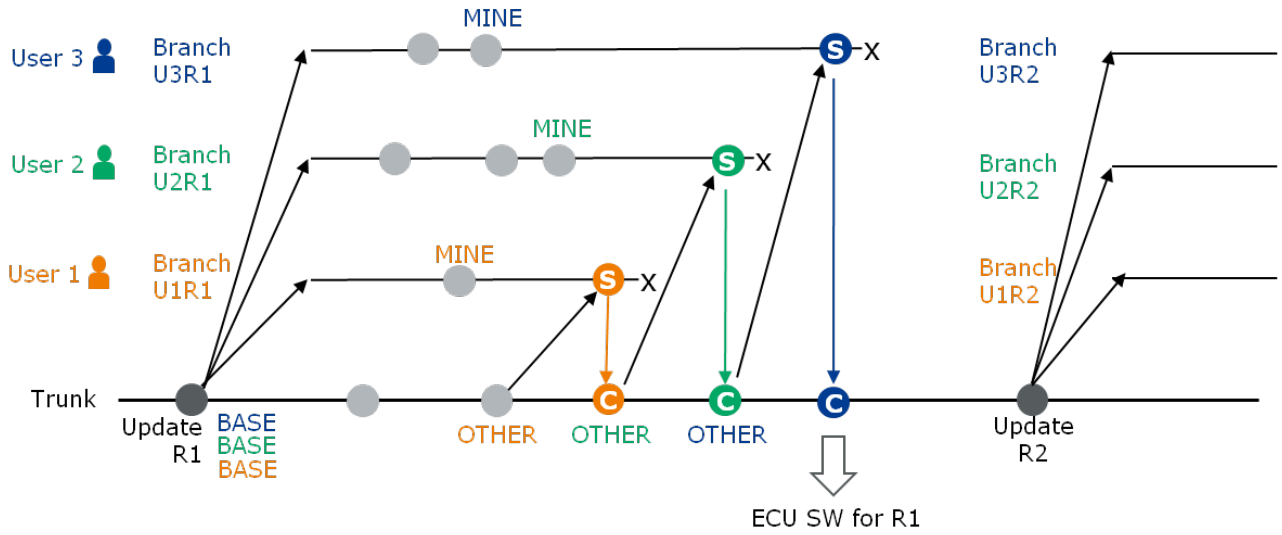


Figure 6-4 Strategy **Central Update on Trunk**

Strategy “Update Each Branch”:

Project updates are performed on the trunk **and** on each user branch. Each user works in his branch and may create revisions. After finishing the development for a particular release, the user branch is merged: sync-merge, then commit-merge. The user branch can “live on”. Immediately after making the commit-merge, the branch is ready for development for the next release, starting with the project update. Special care has to be taken: sync-merges and commit-merges are only allowed with branches on the same release level.

Example: User1 has only minor modifications for R1. He commits his changes early and starts with R2 development with the project update on his branch. After that point, User1 must not make any sync-merges since the trunk is still on R1. The other users continue working on R1. Later on, User2 makes a sync-merge to get the modification of User1, and the commit-merge. Finally, User3 makes the sync-merge to get modifications of User1 and User2, and the commit-merge. Now, the R1 is being released. User2 may decide to make another sync-merge after that to get the modifications of User3 for R1 before he starts with

R2 development. The trunk is being updated to R2 meanwhile. After that point, User1 is allowed to make sync-merges and commit-merges to the trunk again.

This strategy does not block the users during the merge phase, but it may lead to more workload since the project update has to be repeated on the trunk and on each branch separately. Furthermore, more sync-merges may be required.

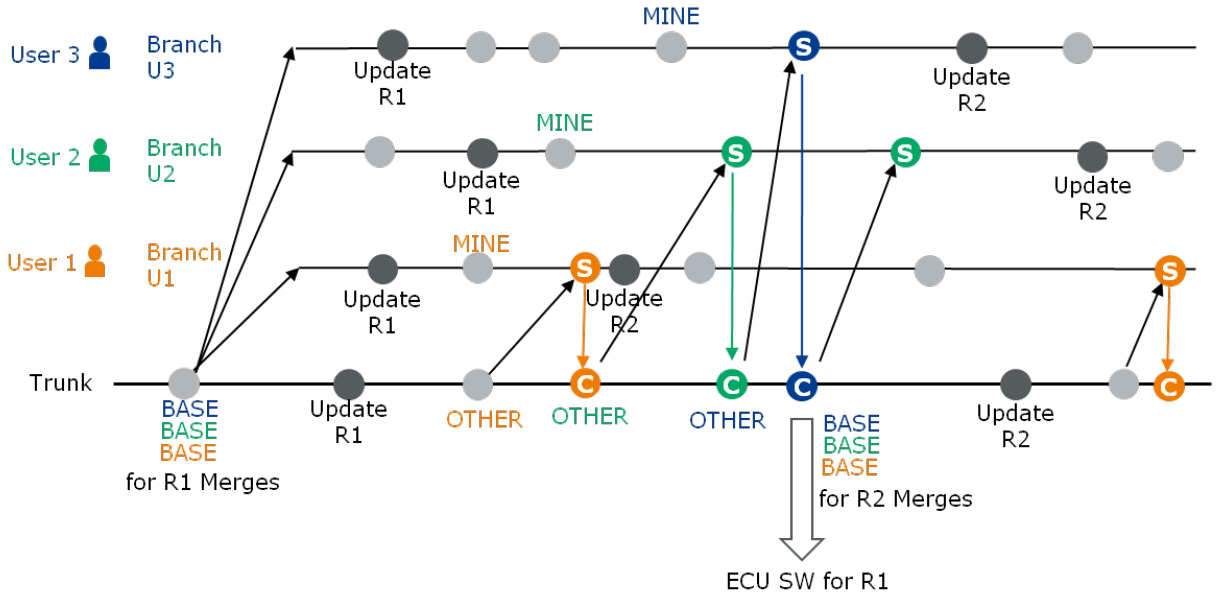


Figure 6-5 Strategy **Update Each Branch**

7 Glossary and Abbreviations

7.1 Glossary

Term	Description

7.2 Abbreviations

Abbreviation	Description
CM	Configuration Management
SDG	Special Data Group
UUID	Universally Unique Identifier
ARXML	AUTOSAR XML
SWC	Software Component
ECUC	ECU Configuration
DCF	DaVinci Configuration File
SIP	Software Integration Package
SVN	Subversion

8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com