

Centralized Lifecycle Management of Model-Based Software Artifacts

Raymond Turin, John Mills, and Jeremy Mangas
SimuQuest Inc

Eileen Davidson, William P. Milam, and Thomas Erickson
Ford Motor Company

ABSTRACT

This paper highlights and discusses the development and deployment of an enterprise-level tool infrastructure that fits into the production build environment of Ford Motor Company. A particular focus is on navigating bottlenecks and pitfalls that arise with the adoption of a model-based software development process. This includes provisions to support centralized data and architecture artifact management (including version control across the lifecycle of the software), support to integrate and manage legacy software artifacts, support to archive and bookshelf development milestones, and last but not least, built-in intelligence to spot potential sources of software defects early in the development stage.

CITATION: Turin, R., Mills, J., Mangas, J., Davidson, E. et al., "Centralized Lifecycle Management of Model-Based Software Artifacts," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 10(1):2017, doi:10.4271/2017-01-0009.

INTRODUCTION

The reception of automobiles in the marketplace is more and more driven by the appeal of software related features, such as infotainment features, advanced powertrain control features, collision avoidance features, etc. Simultaneously, it has become commonplace to develop software artifacts in decentralized locations. As a consequence, embedded automotive software systems have become increasingly complex and the proper management of software artifacts over the lifetime of a vehicle is increasingly difficult. However, recalls are costly, and hence it is imperative that the developed software is error free, despite its complexity and despite coordination challenges.

These challenges give rise to the need for enterprise-level tools to support a unified and guided software development process. This process is commonly formulated around the principles of a model-based software development paradigm. Within this development framework, issues that require specific attention include the management and development of shared model artifacts, the management and specification of shared data definitions, the management and enforcement of uniform coding conventions (e.g., naming conventions, use of units, specification of variants, etc.), and the verification of data integrity between all software artifacts, including the seamless integration of legacy artifacts for a given product release.

This paper addresses these challenges and introduces the concept of centralized artifact management within an automotive production environment.

The paper is organized as follows: First the problems associated with current development practice are highlighted. Then a tool-based development framework is introduced that is focused on promoting innovation rather than circumspection. The paper concludes with a summary and some thoughts for future developments.

CURRENT SITUATION

The emergence of software as a key factor for automotive product competitiveness requires a software development process that fosters rapid deployment of innovative features on the one hand while simultaneously preventing the introduction of errors on the other hand. These are naturally competing objectives. In a traditional software development environment, the introduction of errors is avoided by adding process guidelines. However, these guidelines often stifle innovation and make it harder to meet timing constraints. To make things worse, the full functionality of the final product can only be evaluated at the end of the development process. At this point, all relevant artifacts (data definitions, code artifacts, etc.) have been developed according to specific requirements and it is extremely expensive to make changes. However, experience shows that changes will inevitably be required when the full functionality of the final product is evaluated for the first time.

The schematized workflow below illustrates a typical build process for a Ford powertrain control software release.

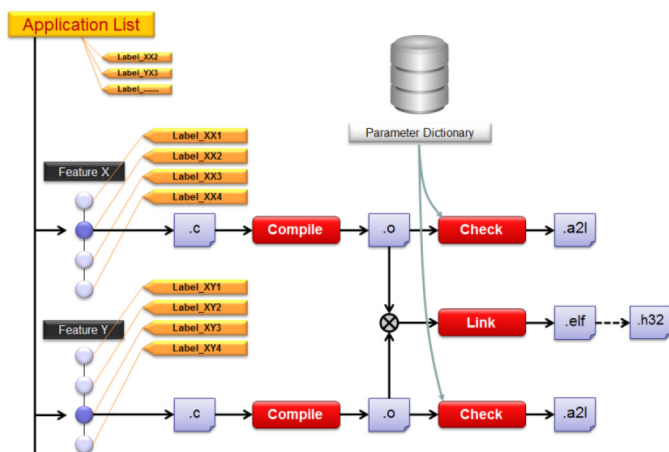


Figure 1. Production control software build process at Ford Motor Company

The release hinges on software artifacts (typically c-code files) that are stored (book-shelved) in the artifact repository of a software configuration management tool. The storing of an artifact involves the creation of a unique label which can subsequently be used to retrieve the artifact from the repository. When an application is built, individual artifacts are selected via an application list which is essentially a list of storage labels (see illustration in Figure 1). The specific composition of labels essentially determines the variant of the built application. The selected artifacts are then compiled and linked. Not surprisingly, it is at the linker stage where problems are looming. This is because the stored software artifacts are not entirely independent as they mutually exchange and reference global data. Whether or not any of the referenced data exists, i.e., whether any of the consumed data objects are defined by exactly one of the artifacts included in the application list, only becomes clear once the linker is invoked. But even if the linker stage is navigated successfully, the final product may still not be flawless. This is because, in a traditional development process, the software artifacts are manually coded and, with the exception of typical unit regression testing, there is little one can do to test the overall functionality early in the development.

Model-based software development has long been heralded as a solution to the many problems associated with the development of complex embedded systems. It has since become the de-facto standard to develop software in most of the automotive industry. Model-based software development resolves some of the traditional issues. For example the functionality of the features, up to the fully integrated system, can be tested in a modeling setting before any code is generated, the introduction of coding errors is minimized due to the use of automated code generation, etc. However, while many problems are resolved, there are numerous emerging issues that need to be addressed. For example (see also illustration in Figure 2):

- how to control data integrity across various generations of a software product
- how to integrate legacy artifacts into a model-based build environment
- how to coordinate collaborative development efforts at different geographical locations
- how to achieve interface consistency throughout the development process

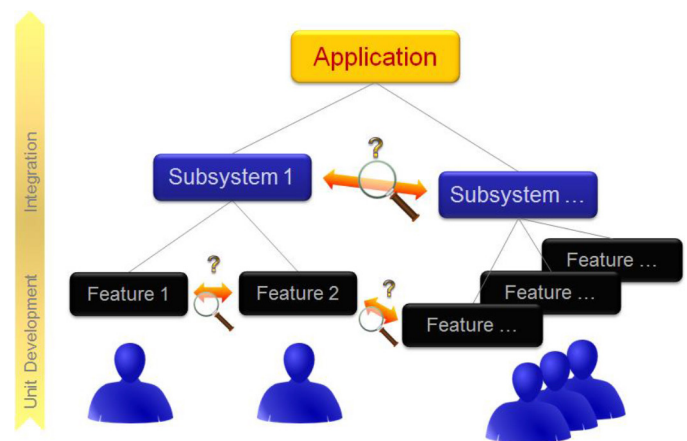


Figure 2. Collaborative development of application artifacts

Many of those issues are currently resolved via ad-hoc methods. For example, data definitions are managed using version controlled excel documents. This is a viable approach in the short term but becomes highly inefficient as the complexity of the software features increases. For example, it is difficult to assess incremental changes of data attributes (data type, initial value, etc.) from version to version, etc., let alone to resolve conflicts on individual data attributes. Furthermore, there is no capability to accurately foretell the effect that a simple data object change has on any existing legacy code artifacts that make use of this data object.

Thus, even with the use of model-based tools, a typical build process currently requires multiple iterations until it can be successfully completed. Failures often emanate from linker errors, which, in turn, are due to the integration of incompatible software artifacts. Diagnosing errors within this environment is not trivial and can significantly delay a launch. Moreover, in order to fix issues quickly, ad-hoc solutions (overriding or sidestepping stipulated process rules) are sometimes adopted that may compromise affected artifacts for the rest of the product lifecycle.

As a consequence, there is a hesitancy to make changes to once functioning product artifacts unless there is an absolute necessity to do so. Clearly, fear from change lest there be a costly issue is not the type of mindset that promotes innovation. But innovation and the associated creation of IP is precisely what is needed in order to obtain and maintain a competitive edge.

Hence, there is a need for a development environment that welcomes experimentation and that is open to new ideas regardless of whether they fit within the current data, architecture or process constraints.

The following section outlines a framework based on intelligent tools that represents a giant step towards such a development environment.

CENTRALIZED ARTIFACT MANAGEMENT

The notion of centralized artifact management means that there is a single authority for the definition, manipulation, and archiving of shared data objects and shared architecture interfaces.

This approach is meant to intrinsically arbitrate any data ambiguity when data is used and edited in the various independent development contexts typically found in an automotive application development program, e.g., the application build process, the modeling environment, calibration, etc. (see illustration in [Figure 3](#)).

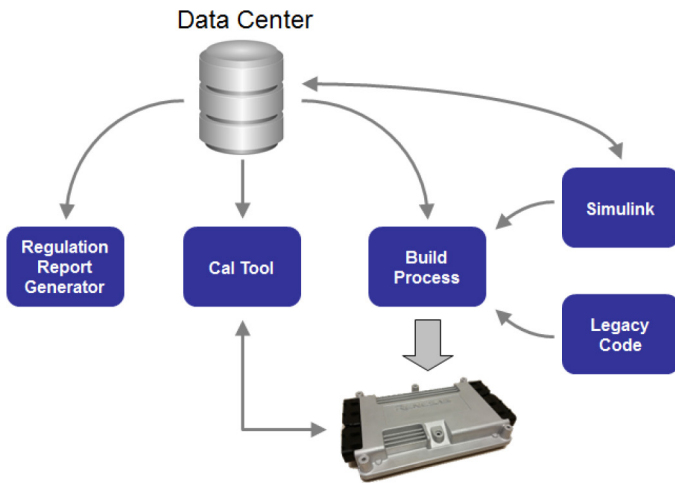


Figure 3. Centralized management of data for automotive applications

Note that while data can be deployed within different development contexts the actual instantiation of the data in any of the contexts strictly follows the definition set forth in the centralized repository. Conversely, if there is a need to change a data definition, the changes must be processed through the data center proper. This process can take different forms. The most natural way is to change the definition in the data center itself. Within a properly designed centralized artifact management framework, those changes will then be automatically propagated back into all usage contexts. However, this natural way may not always be convenient. Any viable centralized artifact management framework also needs to support work flows where data is changed locally (e.g., for optimization purposes during controller calibration) and, once the changes are complete, pushed back into the centralized repository.

The entire idea of centralized artifact management is inspired by the objective to preserve the integrity of data and build artifacts not only throughout the development process but also throughout the product lifecycle. The single authority data center guarantees that any of the data and architecture definitions at any given time in the lifecycle history of the product can be uniquely recreated. As a corollary, this permits the detection of data or interface inconsistencies when a product release requires the integration of artifacts from different time points in the product lifecycle (this is important, e.g., when upgrades to earlier product releases become necessary).

The main objective behind centralized artifact management is to facilitate an agile development process that promotes innovation (i.e., arbitrary changes to selected feature artifacts) but does not compromise the functional integrity of the overall architecture. The agile process is meant to facilitate the detection and elimination of errors at an early development stage while ensuring consistency between all involved artifact interfaces throughout the development process.

A key prerequisite towards this end are tools with built-in intelligence that monitor changes on the fly and report any violation of basic development rules. The use of built-in intelligence to continuously monitor development rules is somewhat related to the notion of continuous integration with automated unit testing (see e.g., [1]).

In many cases this idea is nothing more than a sophisticated, programmatic extension of traditional efforts to validate artifact integrity. Traditional efforts typically define guidelines and rules that need to be checked by human beings. While human beings certainly have many great qualities, the accurate, repetitive processing of rigorous code checks is not one of them. Thus, adding process to eliminate software deficiencies may look fine in theory but, in practice, usually does not yield the desired results. Incorporating rule checking into the tool simply replaces the weak link in the traditional approach with the infallibility of computer logic.

The centralized artifact management framework discussed subsequently involves UniPhi [2], a server-based data and architecture management tool suite in combination with the Matlab/Simulink [3] modeling environment. The UniPhi server is based on a version control backend (subversion). This backend registers each change to the definition of an individual data object as an individual versioning entry. A user working within one of the supported development contexts connects to the server via a UniPhi client. The client supports standard version control operations (checking out, committing, updating, diffing, merging, etc.) as well as the navigation of the change history for each of the version controlled data artifacts. The client tool also supports all aspects of creating, editing, and deleting data artifacts. To access archived artifacts, a user creates a project and checks out a local copy of the needed artifacts. While the described framework represents a stand-alone tool solution, it also includes extensions that tie into the Matlab/Simulink modeling environment. To support the propagation of changed definitions into the implemented feature models, the UniPhi tool suite has a dedicated Simulink library block-set that facilitates the deployment of eligible data within a particular model scope (see [appendix A2](#)).

Roles and Responsibilities

To support program development at an enterprise level, i.e., in an environment where different teams at different locations work in a collaborative effort, it would make sense to designate architecture-specific and implementation-specific roles and organize engineering teams accordingly (see [Figure 4](#)).

In such a scenario, the responsibilities of an architecture design team (consisting of Architecture Engineers, AE) include the definition and management of the architecture artifact interfaces and the responsibilities of the feature implementation team (consisting of Feature Engineers, FE) include the proper implementation of the required feature functionality for a given design artifact while conforming to the interface constraints defined by the AE. Within the centralized artifact management framework, Architecture Engineers will be working with specification artifacts and Feature Engineers with implementation artifacts.