

SPLide: An Integrated Development Environment Supporting Software Product Lines

Ockhyun Paek[1(✉)], Taeho Lee[1], and Jin-Seok Yang[2]

[1] Weapon System Software Reliability Technology Division, Agency for Defense Development, Yuseong, P.O. Box 35, Daejeon 34186, Republic of Korea
{ohpaek, thlee}@add.re.kr

[2] SPID, 18F, 145, Gasan digital 1-ro, Geumcheon-gu, Seoul 08506, Republic of Korea
edward@espid.co.kr

Abstract. SPLide is an integrated development environment for applying software product line to Korean weapon system software development. SPLide supports feature-oriented software development lifecycle. Development of the weapon system software has been made based on a single software development process. Engineers reuse some of the code, common modules and algorithms from previously developed software to increase productivity. However, legacy software is not designed for reuse and cannot be effectively reused. This necessitated strategic reuse, and we developed a tool to support systematic software reuse by applying software product line engineering. In this paper, we propose an integrated development environment that can apply feature-oriented software product line engineering to efficiently develop various weapon system software.

1 Introduction

Weapon systems are generally developed in the form of assembling and integrating dozens of components into one system. Each component can be installed and operated in a variety of similar weapon systems for various purposes through addition, modification, and improvement of functions. Therefore, there exists software that can be commonly used according to the functions of target systems.

Conventional weapon system software has been developed through a process of requirements analysis, design, implementation and testing of a single system without considering systematic reuse. Engineers have partially reused existing design, implementation, and testing artifacts. However, in this way, it is difficult to build up reusable assets in similar areas in the future. This is because a single software development process does not consider the diversity that can occur in terms of customer, environment, infrastructure, and so on.

The SPL platform is the foundation for developing one or more applications with diversity. Tool support is necessary for SPL platform development for various weapon

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11623, pp. 338–348, 2019.
https://doi.org/10.1007/978-3-030-24308-1_28

system domains. There are several studies related to the SPLE methodology and the tools that support it [1–4, 9, 10 and 11]. We developed weapon system SPL processes and tools to comply with the weapon system software development process [5] and to expand with various SW development tools. SPLide is a tool to support all phases of weapon system SPLE processes.

Section 2 describes the components and functions of SPLide, and Sect. 3 introduces the application of SPLide through a simple GCS project.

2 SPLide

We first introduce the overview of the tool and explain how it supports SPLE process. Then, each of modules comprising the tool is explained in detail.

2.1 Overview of SPLide

SPLide supports all phases for weapon system SPLE processes. SPLide operates in a way that the common SPL Repository where assets are managed is shared by various stakeholders. SPLide consists of sub-tools Domain Engineering tool, Repository & Management Tool and Application Engineering Tool as shown in Fig. 1.

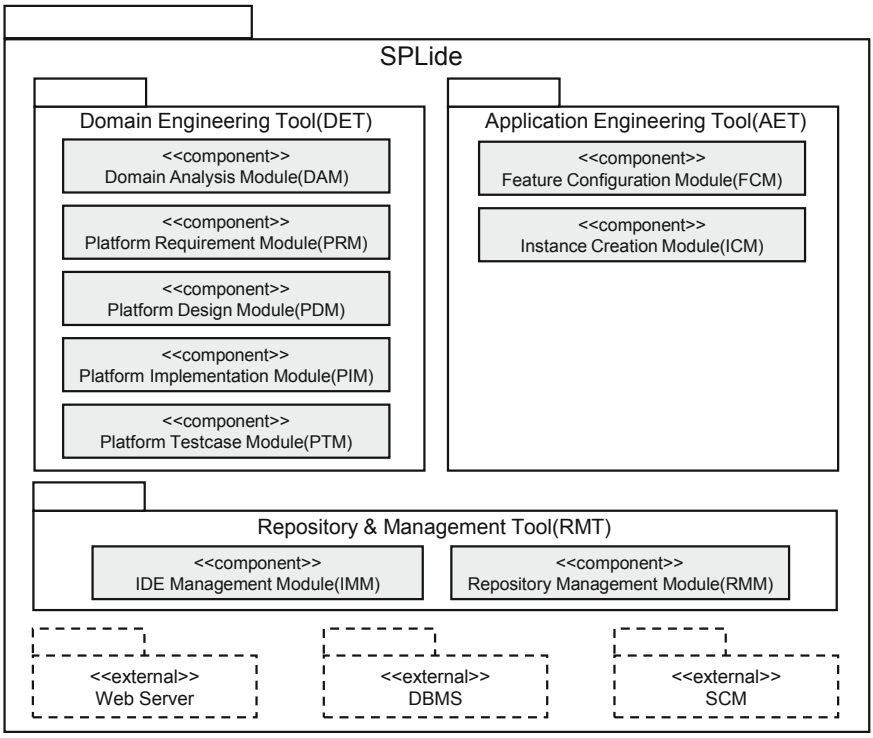


Fig. 1. Logical elements of SPLide

More specifically, two additional elements can be identified. The first is a plug adapter for interfacing with the weapon system software development tool and the second is a platform model library for handling asset data. These two elements are not run independently, but are executable by an interoperable software development tool. For example, IBM Rhapsody's SPLide Plug Adapter has been developed as a plug-in for IBM Rhapsody and can only be run by Rhapsody.

2.2 Weapon System Software Product Line Engineering Process

As described above, existing weapon system software development processes existed only in a single software development process. We define software assets that are constructed to be reusable by separating commonality and variability as 'SPL platform (simply referred to as platform)'. We defined the weapon system SPL process, which is to build SPL platform and to develop software using the platform. The weapon system SPL process is defined based on ISO/IEC 26550 reference model [7] and existing weapon system software development process [5]. The domain engineer and application engineer follow this process to create domain assets and develop SPL platform-based software.

The weapon system software product line engineering process consists of three sub-processes:

Domain Engineering: It consists of the activities to create reusable platform assets to be used in the application engineering process. The commonality and variability of domains are identified using features [8]. In the domain analysis phase, products and features to be included in the product line are derived. The product-feature matrix is created by evaluating the maturity, reusability, and technical feasibility of the feature. This matrix is used to model the variability of the feature. In the platform requirements analysis phase, platform requirements is derived based on feature model and domain knowledge. The domain engineer maps the variability and variable elements within the requirements to the features of the feature model. In the subsequent steps, the developed products can be classified into common/variable assets by associating them with features of the feature model.

Application Engineering: It consists of activities to develop various applications using platform assets. In the software requirements analysis phase, the engineer selects the features to include in the application. Based on the selected features, variation points and variants are determined based on each variability model of development stage, and application artifacts are generated from platform assets. The output of each stage consists of instances derived from platform assets and application specific artifacts.

Asset Management: It consists of activities that manage the change and traceability of assets that will be used in domain engineering processes and application engineering processes.

Figure 2 shows the detailed activities of each engineering process and their relationships. Domain engineering artifacts and application engineering artifacts are stored in an SPL repository. Artifacts include domain models, common domain requirements,

architectural design elements, reusable software components, and application specific artifacts. SPLide supports domain engineering, application engineering, and asset management activities efficiently.

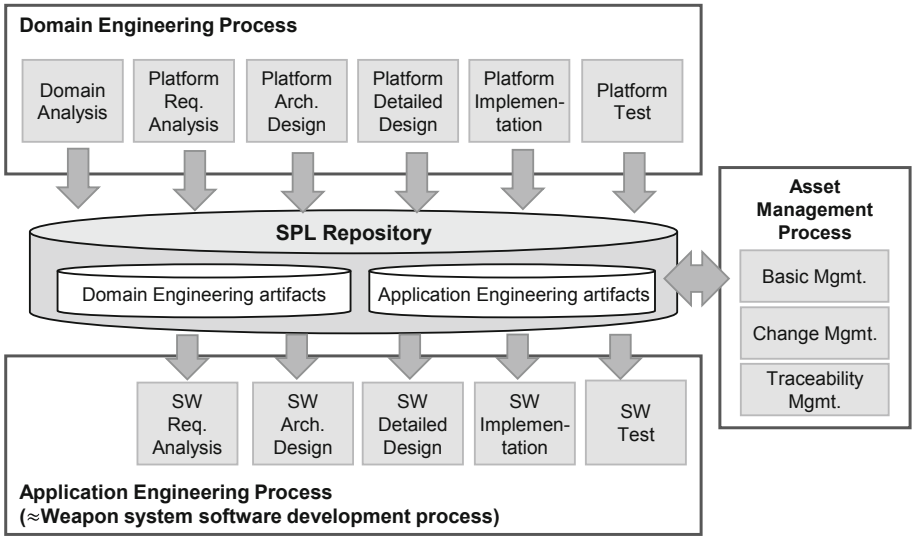


Fig. 2. Weapon system software product line engineering process

2.3 Tools Comprising SPLide

2.3.1 Domain Engineering Tool

The domain engineering tool is a tool that supports the activities of the domain engineering process. It is composed of Domain Analysis Module (DAM), Platform Requirement Module (PRM), Platform Design Module (PDM), Platform Implementation Module (PIM) and Platform Testcase Module (PTM). Each module uses an external tool plug-in for interoperability with software development tools and uses the SPL repository management tool to permanently store and manage platform assets produced through domain engineering activities.

The Domain Analysis Module includes a feature modeling tool to model commonality and variability within a domain. Domain engineers can use feature modeling tools to add/modify/delete features and model relationships between features. It manages the description of features and attributes information of features, manages dependencies between features such as require or mutex through composition rules between features, and examines contradictions that can be caused by composition rules. Feature modeling tools provide feature model check-in/check-out capabilities in the SPL repository.

The Platform Requirement Module (PRM) manages platform requirements. The platform requirements model includes the traceability of the platform requirements specification and the feature model. Engineers use existing requirements management tools to create platform requirements and edit requirements models through plugs

provided by SPLide. As shown in Fig. 3, PRM provides the PRM Plug for linking with various requirements tools, and the Plug provides a plug adapter according to the connected tool. Therefore, plug adapters dependent on the associated tools are needed. The PRM Plug provides two plug adapters, one plug adapter for linking with MS-Excel's document template to specify the platform requirements model and the other plug adapter for connecting with SILKLOAD RM [6], a requirements management tool. PRM provides a web-based platform requirement model management service and can be managed to have valid variability through feature consistency check function.

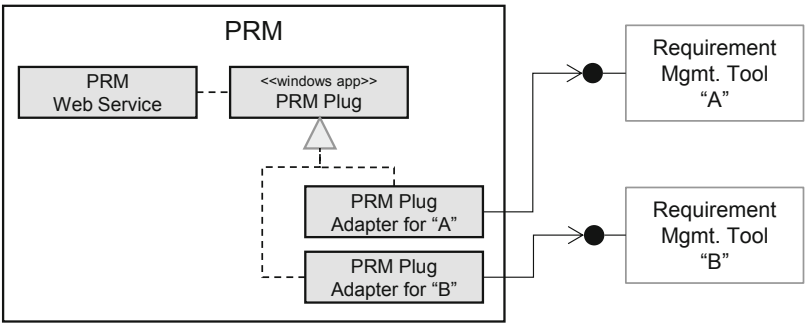


Fig. 3. Plug adapter architecture for interfacing software requirement management tool

The Platform Design Module (PDM) manages the platform design model including variability. The platform design model includes variability information defined by the design information defined by the software design tool and the traceability of the feature model (See Fig. 4).

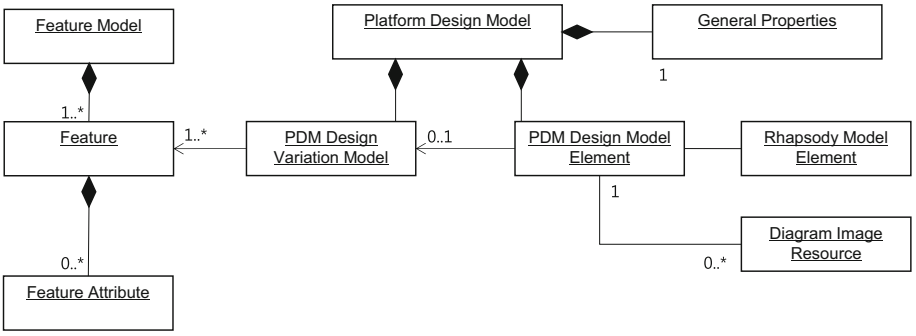


Fig. 4. Conceptual architecture of platform design model

Similar to PRM, PDM provides PDM Plug for linking with various design tools, and Plug provides plug adapter according to the connected tool. The PDM plug provides a plug adapter for IBM Rhapsody. The structure of the plug adapter is similar to that of Fig. 3. Engineers can design software using familiar design tools and establish

relationships with features through adapters optimized for design tools. PDM verifies the validity of the variability defined in the platform design model through syntax checking and consistency checking.

The Platform Implementation Module (PIM) manages the platform implementation model. The platform implementation model consists of the management information of the source code created using the software implementation tool and the variability information defined by the traceability establishment of the feature model. That is, the platform implementation model does not include the actual source code file. The PIM plug is integrated into the code development tool to support activities that implement the source code variability. It provides an environment that extends Eclipse CDT for a more convenient implementation. Engineers can implement variability with PIMs plugged into familiar environments and establish relationships with features. PIM traces variable code corresponding to a particular feature and provides consistency checking for the features used in the implementation.

The Platform Test Case Module (PTM) manages the platform test case model. It consists of traceability information between test cases and feature models defined using software testing tool. Like the PRM and PDM, it provides a plug adapter for linking with the testing tool. The PTM plug provides two plug adapters. Controller tester, a unit test tool, and SILKLOAD TestManager, a system level testcase management tool. The engineer can use the adapter to set the variability information through linkage between test cases and features. The PTM checks whether the user-set variability model is consistent.

2.3.2 Application Engineering Tool

Application Engineering Tool supports the activities of the application engineering process. It uses SPL repository management tools to validate feature configurations based on feature selection and to apply the SPL platform for application instantiation. Engineers can use external tool plug-ins for interfacing and linking with software development tools needed for application requirements analysis, application design, application implementation, and application testing.

The Feature Configuration Module (FCM) provides the ability to select features and feature attributes that remove variability from the feature model. The FCM validates the selected feature configuration and stores the results. Validation includes violation of feature composition rule, whether feature attribute value is properly set, and so on. In addition, it provides the ability to identify differences between different feature configurations.

The Instance Creation Module (ICM) selects a feature configuration and creates an application instance from domain artifacts. Engineers can create a requirement model, a design model, a source code model, and a testcase model linked to a feature as application instances. The generated instance is stored in a specific location, and then a consistency check is performed for each instance.

2.3.3 Repository & Management Tool

The SPL Repository & Management tool (RMT) is a tool for the permanent management of development artifacts and SPL process libraries produced by domain engineering tool and application engineering tool. RMT manages variable points, variable elements, and feature traceability information of development artifacts.

3 Pilot Project

We conducted a pilot project to verify the appropriateness of the proposed SPL process and tool functionality. The domain of the pilot project is the Global Control System (GCS), which controls the unmanned aircraft on the ground. In the Pilot project, the extractive approach was applied because there were products developed by applying the single software development method. After developing the GCS SPL platform through the domain engineering process for the GCS domain, applications were created based on the developed SPL platform.

For the GCS software product line scoping, we created feature matrix by deriving the features for five GCS including the existing product. We modified the method of [1] to determine the GCS product line scope. We determine whether product lines are included in the derived features and model commonality and variability using the feature modeler. Figure 5 shows some of the feature models created with the feature modeler. The GCS feature model consists of total 87 features, 38 mandatory features, and 39 optional features.

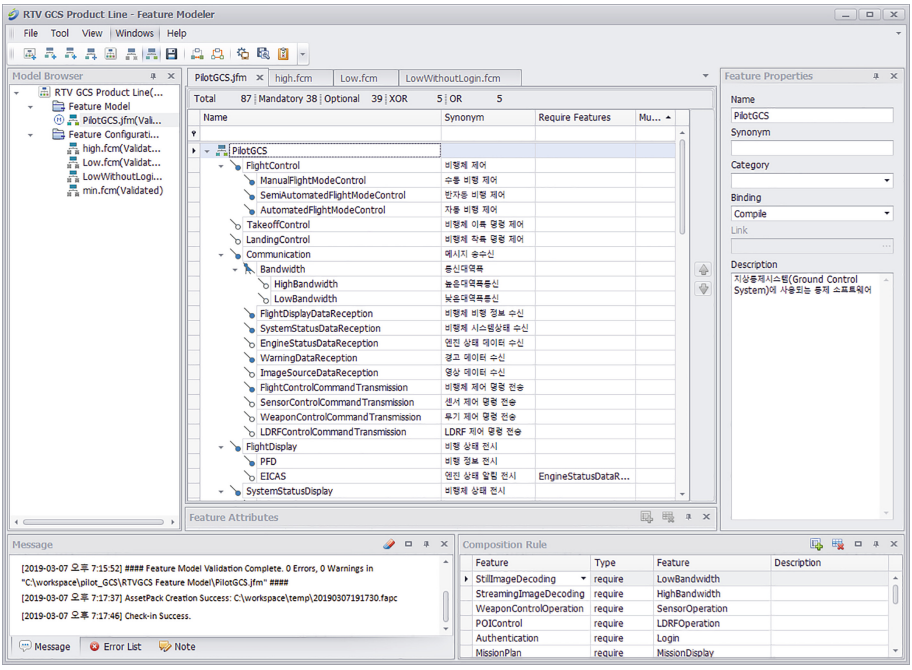


Fig. 5. Feature model of GCS product line

After feature modeling, we analyzed common requirements and variable requirements within the GCS product line. Figure 6 shows the addition of variability to the requirement using the excel adapter. The engineer simply runs the Excel adapter in

Excel and assigns features to requirements. RMM allows multiple engineers involved in the project to check-in/check-out SPL platform artifacts, including variability.

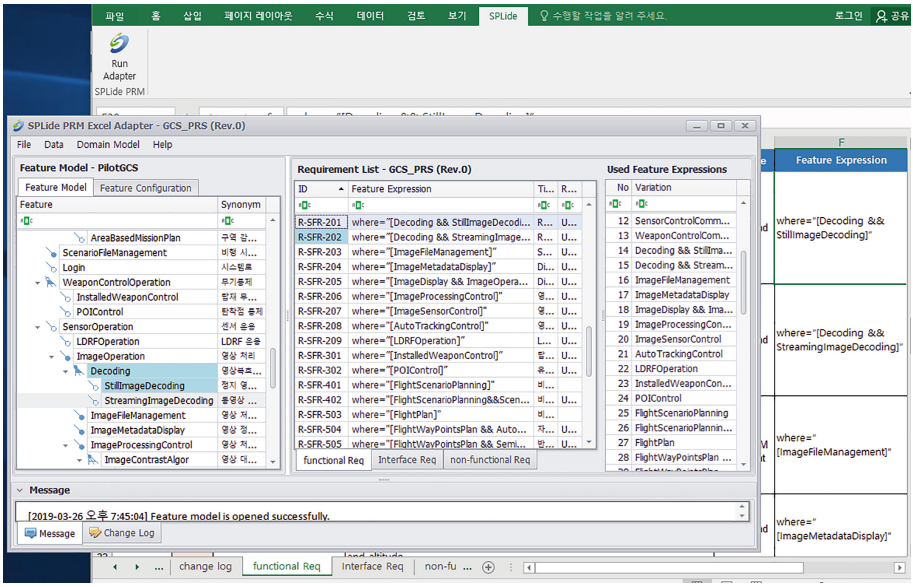


Fig. 6. Platform requirements variability management using excel adapter

Platform design, implementation and test case artifacts are developed in a manner similar to platform requirements. Artifacts were traced to the feature using the adapter of each tool.

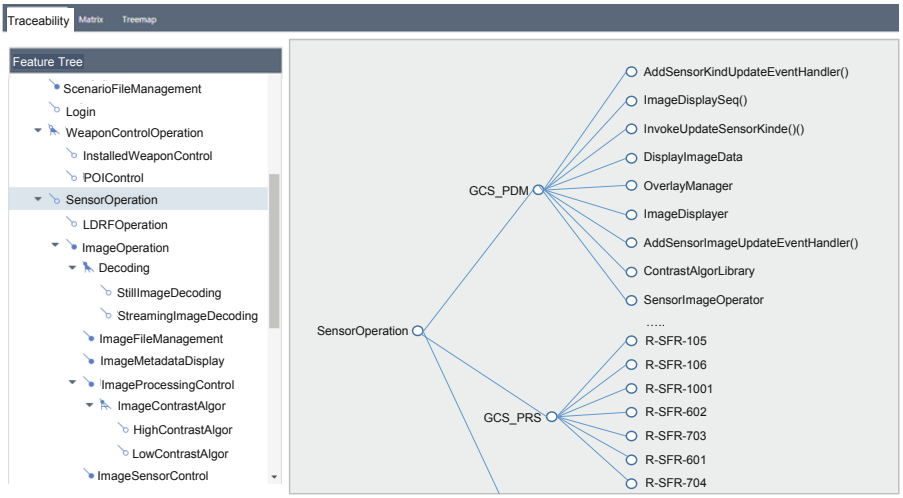
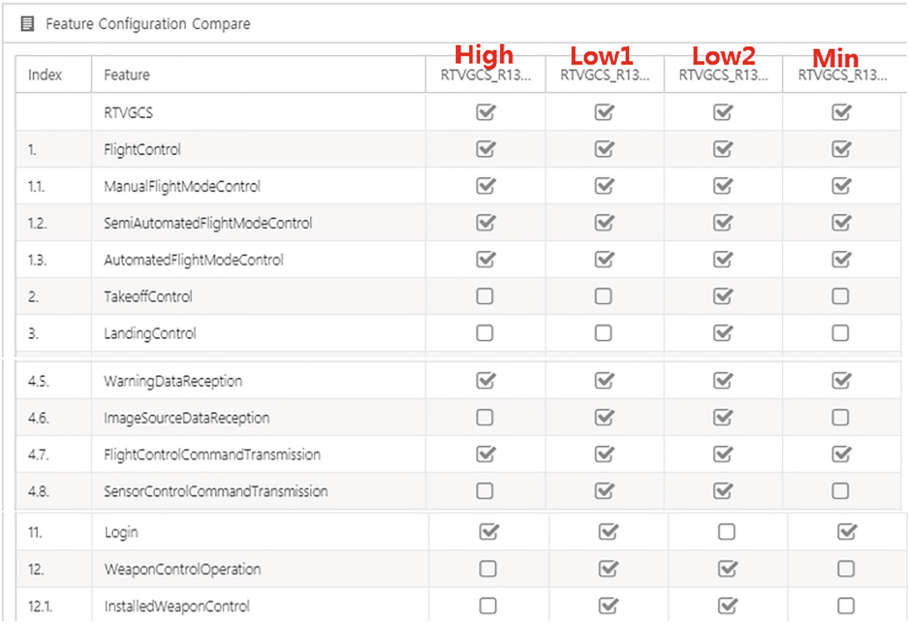


Fig. 7. Feature traceability graph for GCS feature model

Figure 7 shows part of the traceability graph of feature-centered artifacts. When an engineer selects a feature in the feature tree, the tool displays the requirements, design, source code, and test cases associated with the feature. Engineers can easily track domain assets associated with a feature through feature traceability graph.

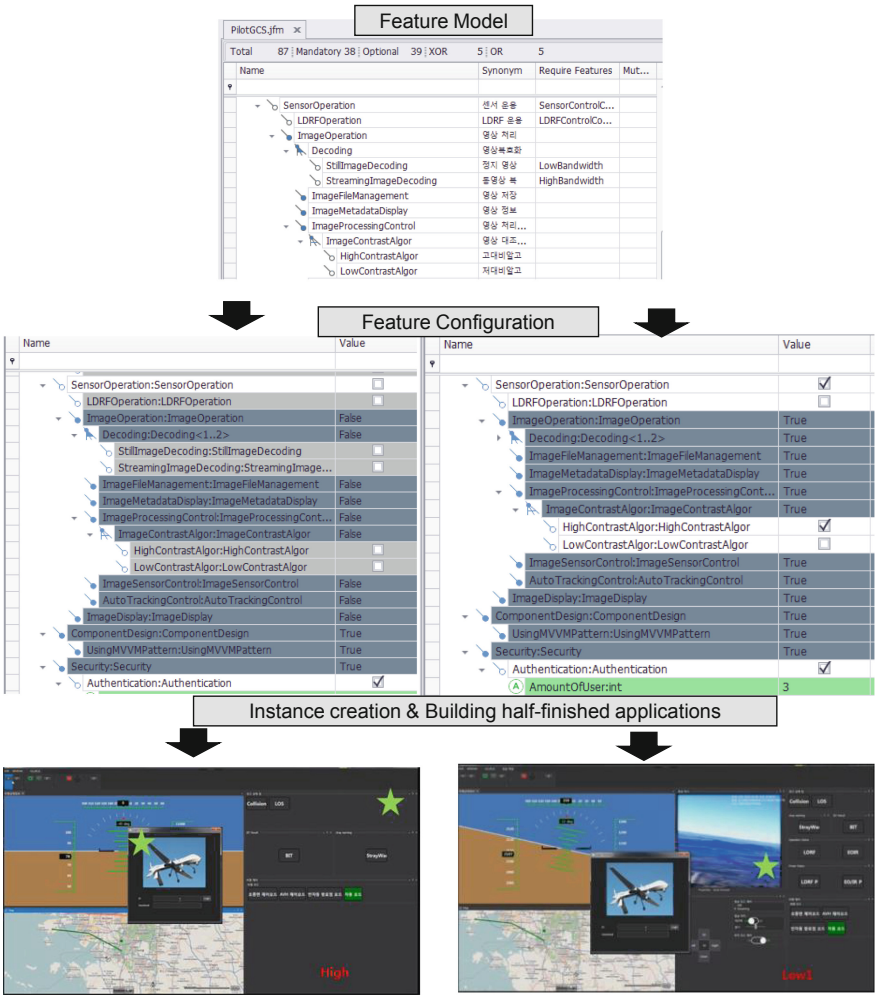
In the application engineering phase, the feature configuration is created by selecting the variable feature of the feature model that meets the requirements of the application and determining the feature attribute value. We created configurations for four applications, as shown in Fig. 8. The tool validates the feature configuration, such as feature composition rule and feature attribute value check.


Feature Configuration Compare

Index	Feature	High RTVGCS_R13...	Low1 RTVGCS_R13...	Low2 RTVGCS_R13...	Min RTVGCS_R13...
	RTVGCS	☑	☑	☑	☑
1.	FlightControl	☑	☑	☑	☑
1.1.	ManualFlightModeControl	☑	☑	☑	☑
1.2.	SemiAutomatedFlightModeControl	☑	☑	☑	☑
1.3.	AutomatedFlightModeControl	☑	☑	☑	☑
2.	TakeoffControl	☐	☐	☑	☐
3.	LandingControl	☐	☐	☑	☐
4.5.	WarningDataReception	☑	☑	☑	☑
4.6.	ImageSourceDataReception	☐	☑	☑	☐
4.7.	FlightControlCommandTransmission	☑	☑	☑	☑
4.8.	SensorControlCommandTransmission	☐	☑	☑	☐
11.	Login	☑	☑	☐	☑
12.	WeaponControlOperation	☐	☑	☑	☐
12.1.	InstalledWeaponControl	☐	☑	☑	☐

Fig. 8. Subset of feature configuration for four applications

Based on the generated feature configuration, instances are created from various platform models. Figure 9 shows examples of half-finished applications by building the source code generated by the instance creation function. In the case of this pilot project, the UI components that make up the screen are also included as part of the domain asset. Generated applications consist only of instantiated artifacts without application specific part. Therefore, it is necessary to reflect application specific requirements, design, and code.



High altitude GCS for relay (High) Low altitude GCS for reconnaissance with Login (Low1)

Fig. 9. Building half-finished applications by creating instances from feature configurations

4 Conclusions

In this paper, we introduced SPLide, an integrated development environment that can be used in the entire software development phases for weapon system software product line. SPLide supports the SPL process to efficiently develop high quality weapon system software based on the weapon system SPL platform. SPLide consists of Domain Engineering Tool, Repository Management Tool, and Application Engineering Tool. In this paper, the function of each tool is described. SPLide uses features to distinguish between commonality and variability. SPLide also maintains traceability between artifacts and features at the development phase. In addition, it provides plug

adapters to work with software development tools. In order to verify the SPLide, we applied the pilot project to provide examples of step-by-step results. We will soon use SPLide to develop a software product line for the new domain. Considering the characteristics of the new domain, we will implement the function of the plug adapter for each step development tool. We will also extend the capabilities for managing platform evolution and for documentation required by the weapon system software development process. We expect SPLide to help engineers effectively manage their SPLs and efficiently develop quality software.

References

1. Bayer, J., et al.: PuLSE: a methodology to develop software product lines. In: Proceedings of the 1999 Symposium on Software Reusability (1999)
2. Beuche, D.: Variant management with pure: variants. Technical report, Pure-systems GmbH (2003)
3. Beuche, D.: Using pure: variants across the product line lifecycle. In: Proceedings of the 20th International Systems and Software Product Line Conference, pp. 333–336. ACM (2016)
4. Krueger, C., Clements, P.: Systems and software product line engineering with BigLever software gears. In: Proceedings of the 17th International Software Product Line Conference, pp. 136–140 (2013)
5. Defense Acquisition Program Administration: Weapon System Software Development and Management Manual (2018)
6. http://nsetec.com/sub/silkroad/silkroad.html
7. ISO/IEC 26550:2013: Software and systems engineering–reference model for product line engineering and management (2013)
8. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study (No. CMU/SEI-90-TR-21). Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA (1990)
9. Kim, J.A., Yang, J.S.: Integration mechanism for software product line development and management workbench. Int. J. Softw. Eng. Appl. **10**(11), 39–48 (2016)
10. Lee, H., Yang, J.S., Kang, K.C.: VULCAN: architecture-model-based workbench for product line engineering. In: Proceedings of the 16th International Software Product Line Conference, vol. 2, pp. 260–264. ACM (2012)
11. Thum, T., Kastner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: FeatureIDE: an extensible framework for feature-oriented software development. Sci. Comput. Program. **79**, 70–85 (2014)