INTERNATIONAL STANDARD

ISO/IEC 26580

First edition
2021-04

Software and systems engineering — Methods and tools for the feature-based approach to software and systems product line engineering

Ingénierie du logiciel et des systèmes — Méthodes et outils pour l'approche basée sur les caractéristiques dans l'ingénierie de lignes de produits logiciels et systèmes

Reference number
ISO/IEC 26580:2021(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

Feature-based software and systems product line engineering ("feature-based PLE" for short) is a specialization of software and systems product line (SSPL) engineering and management that is described in ISO/IEC 26550. ISO/IEC 26550 describes a very generalized approach to SSPL, focusing on the benefits of exploiting a common platform of reusable assets for a product family. Each organization that adopts SSPL under ISO/IEC 26550 is free to do so using their preferred techniques and methods.

What is the motivation for creating a standard for a specialization of SSPL? As the SSPL field has matured and achieved widespread attention in the industry, a specific and repeatable approach to SSPL has emerged that takes advantage of commercial off-the-shelf industrial-strength tools and technology, along with robust best practices for methods and processes, that automate and formalize many of the processes in domain and application engineering. The result is that less upfront analysis, design, and implementation effort is required prior to gaining the benefits from the approach.

While SSPL in general provides significant benefits, it also requires a significant investment of time and effort to adopt and to ultimately achieve those benefits. The feature-based PLE specialization is a more narrowly defined solution that can be supported by off-the-shelf tools and methods, which has resulted in lower investments when an organization adopts SSPL. Feature-based PLE embodies lessons learned about SSPL practices that have been shown to provide some of the highest benefits and returns (see, for example, References [2] and[8]).

This document provides a reference model consisting of an abstract representation of the key technical elements, tools, and methods of feature-based PLE. The predominant specializations of general SSPL that characterize feature-based PLE are:

a) a mapping from features to asset variation points that is sufficient to drive a fully automated configurator that produces assets specific to member products;

b) a methodological shift of all design and implementation effort, change management, and configuration management to domain engineering, so that application engineering is reduced to automated configuration of member product instances and testing of configured member products and member-product-specific assets.

This document embodies a distinct separation of concerns between the feature-based PLE technology providers and feature-based technology users. For each of these stakeholder concerns, the scope of this document is to define only what is necessary and sufficient to enable feature-based PLE practice. For technology providers, this imparts flexibility in how the necessary and sufficient technical capabilities are provided, as well as the opportunity to offer more expansive capabilities that are possible in an ideal solution. For technology users, this provides flexibility to select among the technology providers and to apply the methods that best match their technical and business objectives for feature-based PLE.

Software and systems engineering — Methods and tools for the feature-based approach to software and systems product line engineering

1 Scope

This document is a specialization of the more general reference model for software and systems product line engineering and management described in ISO/IEC 26550. The specialization defined herein addresses a class of methods and tools referred to as feature-based software and systems product line engineering, or feature-based PLE, which has emerged as a proven and repeatable product line engineering and management (PLE) practice supported by commercial tool providers.

This document:

— provides the terms and definitions specific to feature-based PLE;

— defines how feature-based PLE is a specialization within the general ISO/IEC 26550 reference model for product line engineering and management;

— defines a reference model for the overall structure and processes of feature-based PLE and describes how the elements of the reference model fit together;

— defines interrelationships and methods for applying the elements and tools of the product line reference model;

— defines required and supporting tool capabilities.

In this document, products of feature-based PLE include digital work products that support the engineering of a system. Some of the artefacts are actually part of the delivered products, while other artefacts can be non-deliverable, such as physical or digital design models.

The intended audience for this document comprises:

— technology providers who wish to provide automated tool support for the reference model and processes described in this document;

— champions within an organization who wish to introduce feature-based PLE throughout that organization;

— IT staff within a PLE organization who will introduce and maintain the necessary technology to support feature-based PLE;

— practitioner stakeholders who will use the provided technology to practice feature-based PLE;

— technical and business managers who will sponsor and direct the methods necessary to practice feature-based PLE;

— university professors, researchers, corporate trainers, and other educators who will create and share pedagogical materials about feature-based PLE and its benefits.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC/IEEE 12207, *Systems and software engineering — Software life cycle processes*

ISO/IEC/IEEE 15288, *Systems and software engineering — System life cycle processes*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC/IEEE 12207, ISO/IEC/IEEE 15288, and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

NOTE 1 For additional terms and definitions in the field of systems and software engineering, see ISO/IEC/IEEE 24765, which is published periodically as a “snapshot” of the SEVOCAB (Systems and software engineering – Vocabulary) database and is publicly accessible at www.computer.org/sevocab.

NOTE 2 Because feature-based PLE is a specialization of the more general product line engineering approach described in ISO/IEC 26550, some of the terminology used herein is noted as a specialization of the terminology from ISO/IEC 26550, with further details provided in Annex A.

3.1
bill-of-features
specification for a *member product* (3.8) in the *product line* (3.16), rendered in terms of the specific *features* (3.4) from the *feature catalogue* (3.5) that are chosen for that member product

3.2
bill-of-features portfolio
collection comprising the *bill-of-features* (3.1) for each *member product* (3.8) in a *product line* (3.16)

3.3
domain supersets
collection comprising the *feature catalogue* (3.5) and *shared asset supersets* (3.18)

3.4
feature
characteristic of a *member product* (3.8) in a *product line* (3.16) that distinguishes it from other member products in the product line

Note 1 to entry: Features can a) express the customer-visible or end-user-visible variability among the member products in a product line, or b) distinguish implementation variability not directly visible to a customer or end user except through non-functional differences such as price, performance, noise, weight, energy and more.

Note 2 to entry: In feature-based PLE, features express differences among member products. A capability or other characteristic common to all member products in the product line is not modelled as a feature.

Note 3 to entry: See Annex A for the definition of this term in ISO/IEC 26550.

3.5
feature catalogue
model of the collection of all the *feature* (3.4) options and *feature constraints* (3.6) available across the entire *product line* (3.16)

3.6
feature constraint
formal relationship between two or more *features* (3.4) that is necessarily satisfied for all *member products* (3.8)

3.7
feature language
syntax and semantics for the formal representation, structural taxonomy, and relationships among the concepts and constructs in the *feature catalogue* (3.5), *bill-of-features portfolio* (3.2), and *shared asset superset* (3.18) *variation points* (3.20)

3.8
member product
product belonging to the *product line* (3.16)

[SOURCE: ISO/IEC 26550:2015, 3.15, modified — The preferred term "application" has been removed.]

3.9
mutually exclusive
alternatives from which at most one is selected

3.10
mutually inclusive
alternatives from which zero or more are selected

3.11
PLE factory
technological, organizational, and business infrastructure and processes to support a *PLE factory configurator* (3.12) producing *product asset instances* (3.14) from *shared asset supersets* (3.18) based on a *bill-of-features* (3.1) for a *member product* (3.8)

3.12
PLE factory configurator
automated mechanism that produces assets for a specific *member product* (3.8) by processing the *bill-of-features* (3.1) for that member product, and exercising the *shared assets'* (3.17) *variation points* (3.20) in light of the *feature* (3.4) selections made in that bill-of-features

3.13
PLE factory development environment
toolset for creating, organizing, assembling, and maintaining a collection of elements in a *feature catalogue* (3.5), *bill-of-features portfolio* (3.2), *shared asset supersets* (3.18), and a hierarchy of a *product line* (3.16) of product lines

3.14
product asset instance
instantiation of a *shared asset* (3.17) specific to a *member product* (3.8), automatically produced by the *PLE factory configurator* (3.12), corresponding to a *bill-of-features* (3.1) for that member product

Note 1 to entry: A product asset instance is analogous to an application asset (ISO/IEC 26550) with the proviso that it is produced by the PLE factory configurator.

3.15
product instances
collection comprising the *bill-of-features portfolio* (3.2) and *product asset instances* (3.14)

3.16
product line
family of similar products with variations in *features* (3.4)

Note 1 to entry: See Annex A for the definition of this term in ISO/IEC 26550.

3.17
shared asset
software and systems engineering lifecycle digital artefacts that compose a part of a delivered *member product* (3.8) or support the engineering process to create and maintain a member product

Note 1 to entry: A shared asset is analogous to a domain asset (ISO/IEC 26550).

Note 2 to entry: Typical shared assets are requirements, design specifications or models for mechanical, electrical, and software, source code, build files or scripts, test plans and test cases, user documentation, repair manuals and installation guides, project budgets, schedules, and work plans, product calibration and configuration files, mechanical bills-of-materials, electrical circuit board and wiring harness designs, engineering management plans, engineering drawings, training plans and training materials, skill set requirements, manufacturing plans and instructions, and shipping manifests.

3.18
shared asset superset
representation of a *shared asset* (3.17) that includes all content needed by any of the *member products* (3.8)

3.19
variant
alternative that can be used to realize a particular *variation point* (3.20)

[SOURCE: ISO/IEC 26550:2015, 3.28, modified — the word "one" at the beginning of the definition has been removed; "may" has been changed to "can"; "particular variation points" has been changed to "a particular variation point"; note 1 to entry has been removed.]

3.20
variation point
identification of a specific piece of *shared asset superset* (3.18) content and a mapping from *feature* (3.4) selection(s) to the form of that content that appears in a *product asset instance* (3.14)

Note 1 to entry: In this document, all features express characteristics that differ among *member products* (3.8), which according to ISO/IEC 26550 would also make every feature a variation point. To avoid this redundancy, this document does not call out features as variation points.

Note 2 to entry: See Annex A for the definition of this term in ISO/IEC 26550. The definition in this document is more specific to feature-based PLE and the *PLE factory configurator* (3.12) approach of producing product asset instances from shared asset supersets.

4 Overview of feature-based product line engineering

4.1 General

This clause gives a brief informational overview of feature-based product line engineering, as a way of introducing key concepts that are important for understanding the purpose and content of this document.

Software and systems product line engineering emerged some time ago as way to engineer a portfolio of related products in an efficient manner, taking full advantage of the products' similarities while respecting and managing their differences. Here "engineer" means all of the activities involved in planning, producing, delivering, deploying, sustaining, and retiring products. Born in the software engineering field in the 1970s and 1980s and based largely on the concept of software reuse, PLE has long been known for delivering significant improvements in development time, cost, and quality of products in a product line[1].

Early attempts to capture and codify best practices recognized a dichotomy between two sides: product content that applies to multiple products and product content specific to a single product. Some referred to the two sides as domain engineering and application engineering, respectively, while others referred to core asset development vs. product development[1]. Application engineering was often said to include creating any content that happened to be used only in a single product, and promoting that content to domain status only if subsequently used in more. Application engineering included the obligation to choose and carry out a production strategy – that is, a way to turn the shared assets into products – that was potentially different for each type of shared asset, and was often manual and therefore labour-intensive and error-prone. Whatever they were called, the two sides stood on roughly equal footing in terms of the effort required to execute.

However, starting in the early 2000s, the advent of industrial-strength and commercially available technology designed specifically to support PLE enabled the rapid emergence of a specialization of PLE practices. This approach is called feature-based software and systems product line engineering[5].

Under this approach, the technology just mentioned allows application engineering, which was so important under the “classic” PLE approach, to shrink to almost nothing. Products are produced through the use of high-end industrial-strength automation that configures the shared assets appropriately for each product. Feature-based PLE explicitly declines to configuration-manage or change-manage product versions. Instead, the shared assets (and not the individual products or systems) are managed under CM (configuration management). A new version of a product is not derived from a previous version of the same product, but from the shared asset supersets themselves. Additionally, any defects are fixed in the shared assets, not the products. The affected products will then be regenerated, into a form suitable for testing and deployment. Since regeneration has a low and fixed cost, it matters very little whether 2 or 200 or 2000 products need to be regenerated. Thus, fixing a defect, making a systematic enhancement, or carrying out any other kind of multi-product change becomes much more economical[4].

4.2 to 4.4 describe a few more of the key ways that feature-based PLE differs from its antecedents.

4.2 Shared assets

Shared assets are the “soft” artefacts associated with the engineering lifecycle of the products, the building blocks of the products in the product line. Assets can be whatever artefacts that are representable digitally and either compose a product or support the engineering process to create a product. Examples include, but are not limited to, the following:

— requirements;

— design specifications and design models for mechanical, electrical, and software;

— software source code and build files;

— test plans and test cases;

— user documentation, repair manuals, and installation guides;

— project budgets, schedules, and work plans;

— product calibration and configuration files;

— data models;

— process descriptions;

— parts lists and mechanical bills-of-materials;

— engineering drawings;

— electrical circuit board and wiring harness designs;

— training plans and training materials;

— skill set requirements;

— manufacturing plans and instructions;

— shipping manifests;

— marketing brochures;

— product descriptions;

— contract proposals.

Shared assets are engineered to be used (shared) across the product line. All of this was true before, but in feature-based PLE, shared assets take the form of supersets, meaning that any asset content used in any product is included. For example, a shared asset of requirements contains all of the requirements across the product line; a shared asset of computer software source code contains all of the source code; and so forth.

The virtue of this approach is that every piece of content for the product line, no matter in which products it is used, is only created, stored, and maintained once. There is no duplication or replication of asset content at all. This elimination of duplication (and elimination of replication of work across duplicates) is where feature-based PLE derives its savings[5].

The shared asset supersets include variation points, which are places in the asset that denote content that is configured according to the feature selections made for the product being built. When a product is built, a statement of the product's distinguishing characteristics – its features – is applied to "exercise" these variation points (that is, cause the content associated with each variation point to be configured in a way to meet the needs of the product).

Configuration capabilities typically include inclusion or omission of the content; selection from among mutually exclusive content alternatives; generation of content from feature specifications; and feature-based transformation of content from one form into another.

This approach enforces consistent treatment of all of the variation points in all of the shared assets; for example, variation points are specified using the same feature-based configuration language. This consistent treatment of variation across all shared assets is a hallmark of feature-based PLE. The result is that a full set of demonstrably consistent supporting artefacts can be systematically produced for each product.

In contrast, consider approaches where each domain asset type has its own variation mechanism. For example, a requirements asset engineering team has embraced a PLE requirements management technique based on tagging requirements in a requirements database with attributes that differentiate feature variations in requirements. Further, the design team has adopted a SysML tool and has embraced inheritance as the mechanism for managing their design asset variations. The software asset development team is using a feature-oriented domain analysis (FODA)[7] feature model drawn in a graphical editor, plus #ifdefs, build flags and configuration management branches to manage implementation variations. Finally, the test team has adopted clone-and-own (i.e. make a copy and take ownership of modifications to the copy) of test plan sections, stored in appropriately named file system directories to manage their test asset variations.

Now consider what would be needed to create a complete PLE lifecycle solution from these unrelated asset variation mechanisms that integrates into a larger business process model. How do the requirements asset attributes and tagged requirements relate and trace to the subtypes and supertypes in the design assets? How do these attributes and supertypes relate and trace to the #ifdef flags, CM branches, FODA features in the software assets, and test asset clone directories? Trying to translate between the different representations and characterizations of features and variations among the domain assets creates convoluted dissonance at the boundaries between stages in the domain asset engineering lifecycle.

To resolve this quagmire of multiple disparate mechanisms for managing product line asset variation, a key aspect of feature-based PLE is consistent and traceable treatment of variation points across all shared asset types, using feature choices as the basis of a common language of variation[1].

4.3 Features

As the name implies, features play a central role in feature-based PLE. Features express differences among products in a product line. A capability or other characteristic common to all products in the product line is not modelled as a feature. A feature is a distinguishing characteristic of a product, usually directly visible to the customer or user of that product, but can also express a distinguishing implementation variability in any phase of the lifecycle that is not directly visible to a customer or end user except through non-functional differences such as price, performance, noise, weight, energy and more. An example is a function that one product can perform that others cannot. The concept of “feature” allows experienced domain experts and systems engineers to employ a consistent abstraction when defining and making selections from a whole product configuration, all the way down to the deployment of components and parts within a low-level subsystem in the architecture[3]. A bill-of-features (analogous to a bill-of-materials, but defining a product in terms of its features rather than its parts) can be the communication vehicle among business, product marketing, and engineering units.

4.4 Automated means of production

At the centre of feature-based PLE is an automated mechanism called a configurator that exercises the shared assets’ variation points to produce configured variants that, together, constitute the artefact set for each of the products in the product line. PLE configurators have been commercially available since the early 2000s.

Previous PLE approaches have always made allowances for automation, but large-scale PLE demands it. The complexities of modern product lines are growing to astronomical proportions. For example, an automotive product line may comprise thousands or millions of products[8]. Automated generation of products is essential. Once this technology is in place, products are instantiated rather than manually created.

5 A feature-based specialization of software and systems product line engineering

ISO/IEC 26550 describes software and systems product line engineering (SSPL), a general approach for efficiently engineering the full lifecycle of digital assets for a family of similar products or systems. The approach derives its benefits from reusing and sharing the digital engineering assets across the products in the family and by systematically managing variations among the digital assets for the different products. A reference model for this overall approach is defined in ISO/IEC 26550. The primary illustration from ISO/IEC 26550 is shown in Figure 1.

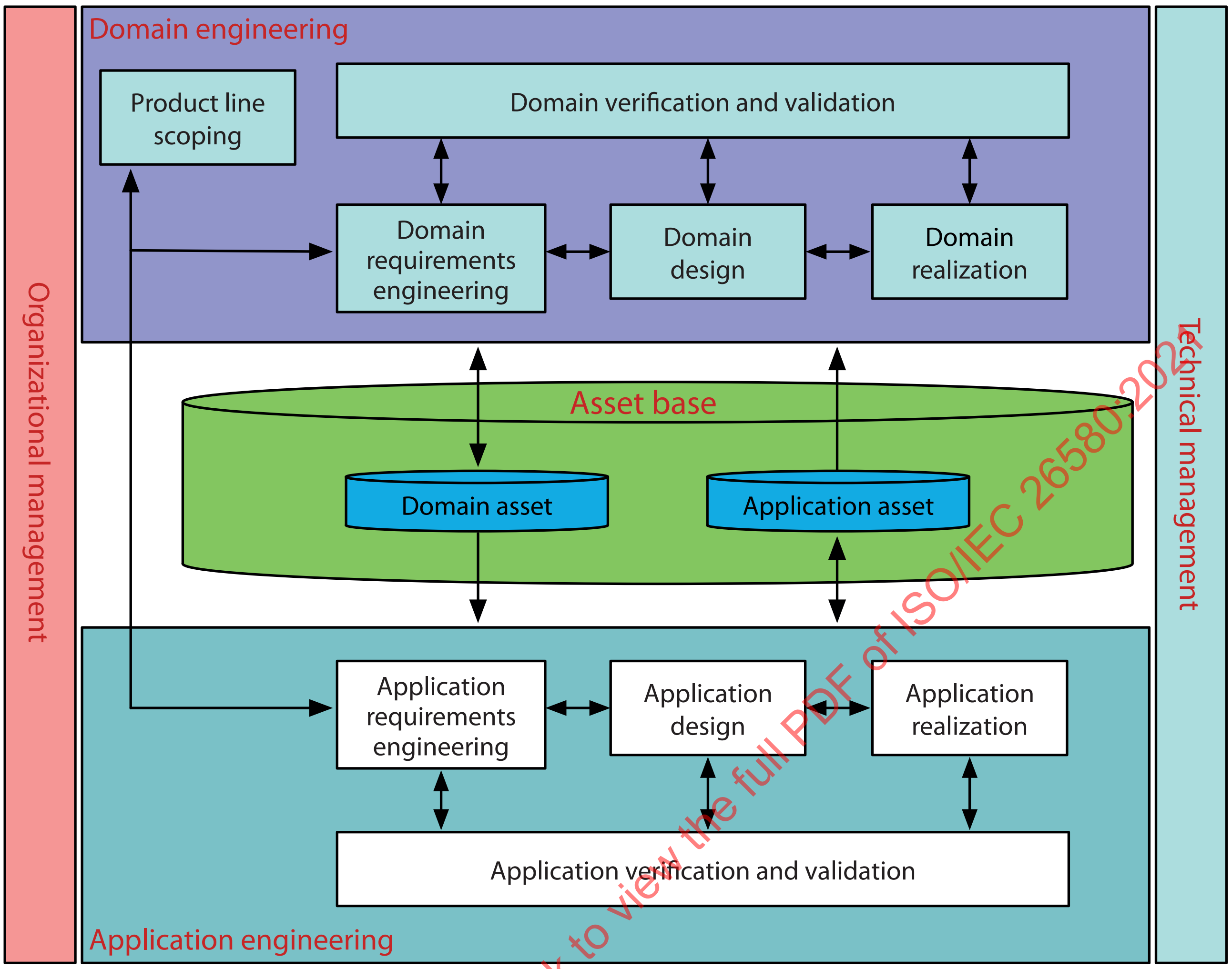

Figure 1 — ISO/IEC 26550 for SSPL engineering and management

This document defines feature-based PLE, which is a specialization of the general SSPL approach defined in ISO/IEC 26550 and illustrated in Figure 1. This clause is provided for users who would like to understand how feature-based PLE is a specialization of ISO/IEC 26550 and is not otherwise a prerequisite for understanding the remainder of this document. The predominant differences and unique characteristics of feature-based PLE are illustrated in Figure 2, elaborated within the context of the general SSPL approach.

a) The asset base, illustrated by the large cylinder in both figures, takes on several characteristics that are specific to feature-based PLE:

— Under feature-based PLE, domain assets, illustrated by the smaller cylinder on the left-hand side, are referred to as domain supersets and explicitly incorporate a formal feature catalogue which has a central role in abstraction and automation in the approach. Features serve as the common language for communicating product line variability among all disciplines and all roles in feature-based PLE. In ISO/IEC 26550 terms, the feature catalogue serves as a variability model that is orthogonal to (instead of being embedded in) all other assets within a product line. Under feature-based PLE, graphical feature models, illustrated by an image of a feature model above the "Feature catalogue" text box in Figure 2, are usually used to model variability.

— Domain assets also include the conventional engineering assets, shown in the systems engineering 'V', enhanced with variation points – places in the assets that vary depending on the abstract feature variations. (Places in the assets that do not vary do not require any such enhancement and are made a part of every member product.) Extending engineering assets with variation points results in shared asset supersets that can be automatically configured based on selected features, to support different products in the product line.

— Under feature-based PLE, application assets, illustrated by the smaller cylinder on the right-hand side, are referred to as product instances and explicitly incorporate a formal bill-of-features portfolio, which is a collection of individual bill-of-features, one for each member product in the product line. Each member product has a unique bill-of-features in which specific features have been selected from the feature choices of the feature catalogue to instantiate the variability offered by the feature catalogue. In ISO/IEC 26550 terms, the variability model afforded by the feature catalogue has been bound into the bill-of-features. The images of three bound variability models above the “Bill-of-features portfolio” text box show how three unique feature-based representations of products have been created by selecting certain feature choices from the variability model shown above the "Feature catalogue" text box in Figure 2.

— Application assets also include the conventional engineering assets, shown as the three systems engineering ‘V’s above the “Product asset instances” text box. No product line variability remains in the product asset instances.

b) The PLE factory configurator, between the domain assets and application assets cylinders, provides the automated configuration of the shared asset supersets into product asset instances for the member products, based on input about feature selections made in a product bill-of-features. This automation is sufficient to eliminate the manual development effort that is traditionally associated with application engineering. Note that the PLE factory configurator in the illustration is placed in a cutout to show that it is separate from the asset base.

c) Domain engineering, illustrated by the "Domain engineering" box across the top, takes on several characteristics that are specific to feature-based PLE:

— Under feature-based PLE, domain requirements engineering, domain design, domain realization, and domain verification and dalidation are all extensions to conventional engineering across the ‘V’ lifecycle of the shared asset supersets. The relations shown by arrows among these items in Figure 1 are the corresponding relations implied among their ‘V’ counterparts in the domain assets cylinder of Figure 2.

— These extensions involve the addition of variation points into the assets across the lifecycle. Beyond that, feature-based PLE has no bias in the engineering tools and methods that are applied in each of the lifecycle stages.

— In feature-based PLE, product line scoping is as described in ISO/IEC 26550. The three aspects of product line scoping are carried out as follows. Domain scoping is as described in ISO/IEC 26550. Product scoping is performed in conjunction with bill-of-features portfolio engineering (described in 8.4). Asset scoping is performed in conjunction with shared asset superset engineering (described in 8.5).

d) Application engineering, illustrated by the "Application engineering" box across the bottom, takes on several characteristics that are specific to feature-based PLE:

— Under feature-based PLE, application requirements engineering, application design, application realization, and application verification and validation are all extensions to conventional engineering across the ‘V’ lifecycle of the product asset instances. The relations shown by arrows among these items in Figure 1 are the corresponding relations implied among their ‘V’ counterparts in each product asset instance in the application assets cylinder of Figure 2.

— Under feature-based PLE, product asset instances are automatically generated by the PLE factory configurator from shared asset supersets and a bill-of-features that defines which features from the feature catalogue are selected for a particular product in the product line. There is no application engineering required on the application assets that are derived from the domain assets.

— The emphasis of application engineering is on verification and validation on the right-hand side of the ‘V’. Manual and automated verification and validation assets are generated by the PLE factory configurator along with other product assets to improve the efficiency and accuracy of testing. If issues are found during verification and validation, the defect fixes and enhancements are done back on the domain engineering side in ISO/IEC 26550 terms, as shown by the feedback arrow from product verification and validation to domain engineering. The arrow goes through the "Technical management" box to denote that some technical organization management processes are involved in dealing with the feedback. Most notably, the change management process (8.10) determines what changes, if any, to the feature catalogue, bills-of-features, and shared asset supersets need to be done based on the feedback. The product asset instances are then regenerated, re-verified, and re-validated;

— Because product asset instances are generated, they can be treated as transient digital data. They can be discarded and regenerated at a later time as needed.

Thus, feature-based PLE brings a methodological shift of design and implementation effort, change management, and configuration management to domain engineering. It reduces application engineering to the automated configuration and testing of configured product asset instances. This is extremely significant since application engineering effort typically dominates the effort in the early generation SSPL approaches.

The variability offered by a product line is represented by feature choices in the feature catalogue of the product line. The variability model provides a centralized, holistic view of variability because all variability is always introduced and managed by means of the variability model. Shared asset supersets implement only the variability specified by the variability model that the feature catalogue represents. They cannot offer specific variability through variation points unless the variability has been introduced and required by the variability model, and a formal mapping exists between the variation points and the specific features in the feature catalogue introducing and requiring the variability.

The strong and explicit correspondence among the feature catalogue, the shared asset supersets, and the bill-of-features portfolio facilitates a variety of analyses to manage the product line. For example, for each feature in the feature catalogue, it is straightforward to find out which member products use it, which shared asset supersets have been designated to implement it, and which product asset instances have been generated to implement it. It is also straightforward to track for each product which features are used in the product and which shared asset supersets are needed to implement the product.

Feature-based PLE can be applied in conjunction with other PLE processes, such as those detailed in ISO/IEC 26550 for product line engineering and management. For example, modification of generated product asset instances is not included as a practice in this document, but organizations that choose to adopt that practice should follow the application engineering methods defined in ISO/IEC 26550.

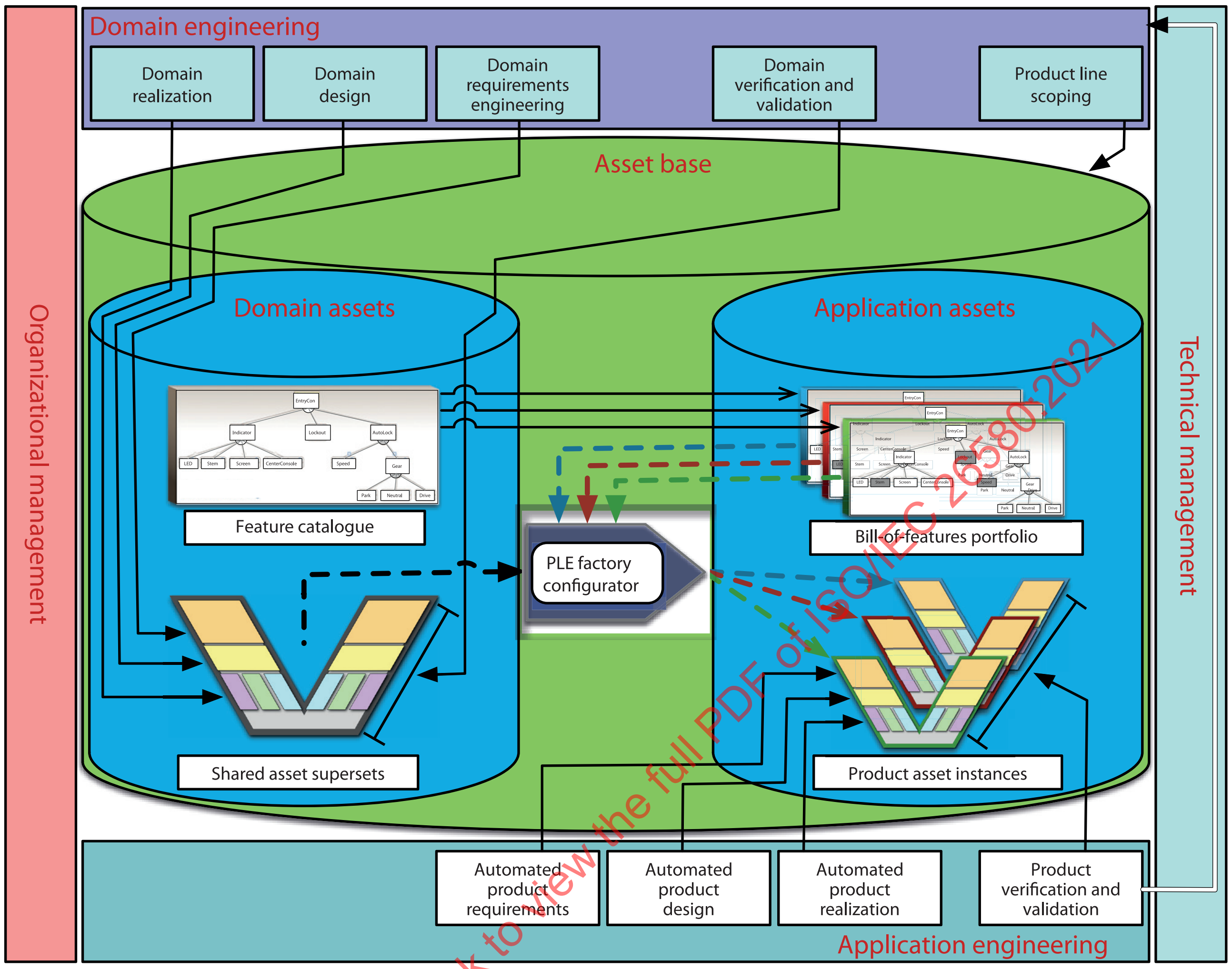

Figure 2 — Feature-based specialization of ISO/IEC 26550

6 Reference model for the feature-based approach to software and systems product line engineering

6.1 General

This clause defines a reference model, illustrated in Figure 3, that forms the basis for executing feature-based PLE practices. The reference model is based on a factory metaphor: The feature-based PLE factory produces digital assets from across the software and systems engineering lifecycle for each of the member products in a product line, analogous to a conventional factory producing physical assets and products.

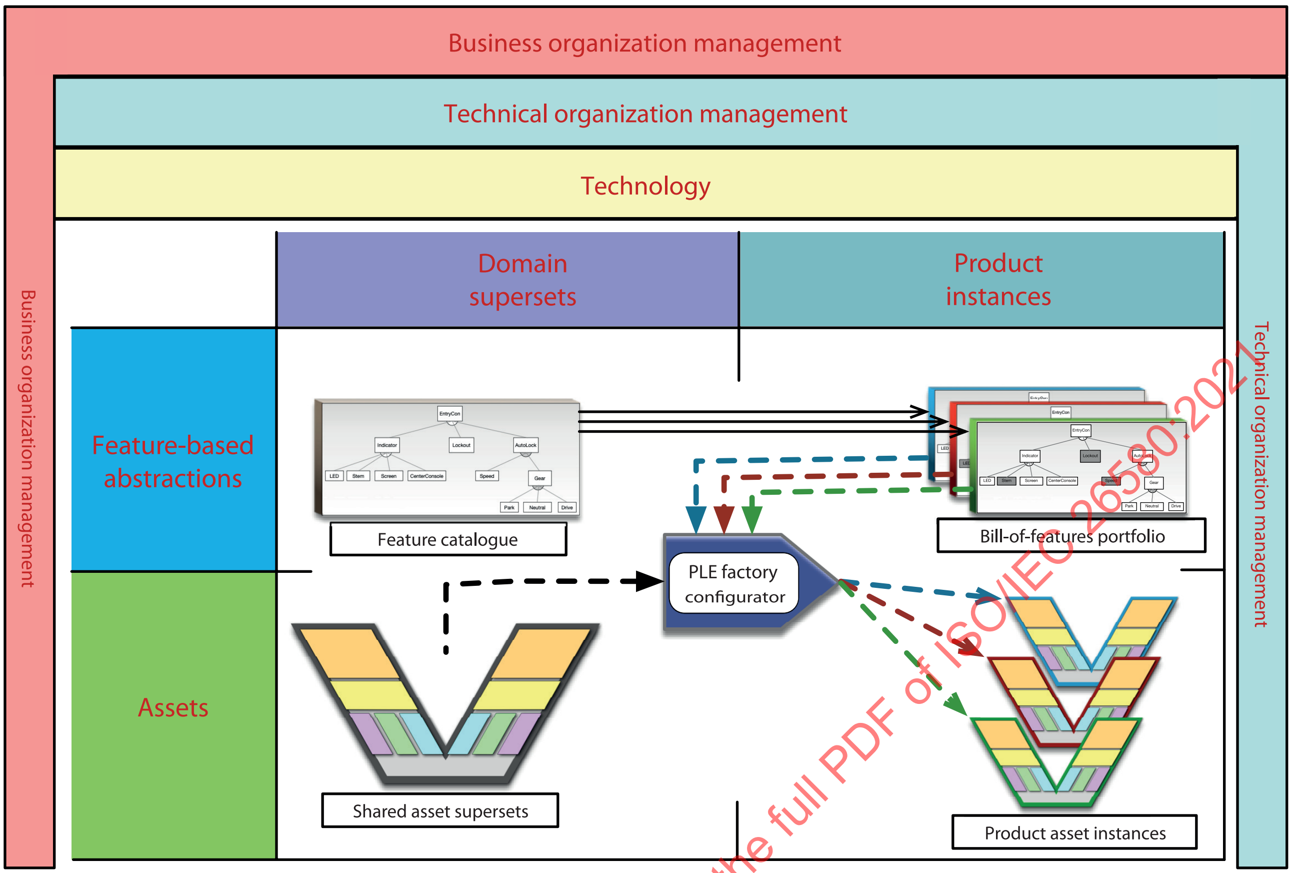
Business organization management
Technical organization management
Technology
Domain supersets
Product instances
Feature-based abstractions
Assets
Feature catalogue
Bill-of-features portfolio
PLE factory configurator
Shared asset supersets
Product asset instances
Business organization management
Technical organization management

Figure 3 — Reference model for the feature-based approach to software and systems product line engineering

Figure 4 through Figure 9 build up the full reference model in stages. Dashed arrows in these figures indicate information flow and solid arrows indicate 'is an instance of.' The corresponding UML class and component diagrams for the key elements and relationships in the feature-based PLE factory reference model are provided in Annex B.

6.2 Key elements of the feature-based PLE factory

6.2.1 General

Figure 4 shows the key elements of the feature-based PLE factory, which are described below.

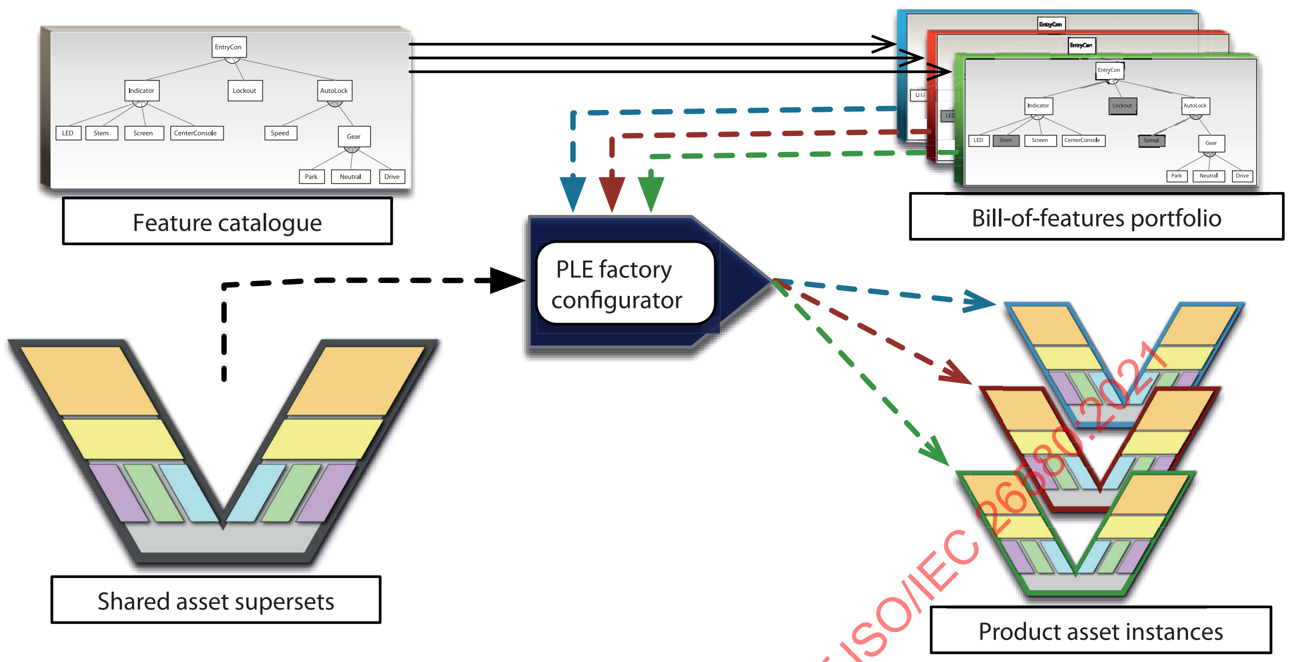

Figure 4 — Key elements of the feature-based PLE factory

6.2.2 Feature catalogue

A product line's feature catalogue is a model of the collection of all of the feature options and feature constraints that are available across all member products in the product line.

A feature is a distinguishing characteristic of a product within a product line[7]. An example is a function that some member products can perform that others cannot. Features are analogous to the choices one makes when purchasing or ordering a product. Features often express the customer-visible or end-user-visible variability among the products in a product line, but can also capture distinguishing implementation variability not directly visible to a customer or end user except through non-functional differences such as price, performance, noise, weight, energy consumption, and more.

The concept of feature allows a consistent abstraction to be employed when defining and making choices from a whole product configuration, all the way down to the deployment of components within a low-level subsystem in the architecture. Features provide the common communication vehicle – a lingua franca – among all stakeholders in the product line, from requirements engineers to testers, from marketeers to executives, from designers to customers.

One of the more powerful engineering techniques for reducing cognitive complexity for engineers in any discipline is abstraction and automation. Complex constructs are encapsulated and hidden behind simplifying abstractions, while automation provides a mapping from the abstractions to the hidden and more complex engineering constructs. Source code compilers and application generators are examples. The feature-based PLE factory serves exactly this purpose. Features in the feature catalogue provide a simplifying abstraction that hides the exponential complexity of possibly hundreds, thousands or even millions of interacting points of variation within the engineering assets.

Feature catalogue engineering, described in 8.3, is the activity that produces and maintains a feature catalogue that captures all of the germane distinguishing characteristics among the products in the product line.

6.2.3 Bill-of-features and bill-of-features portfolio

A bill-of-features is a specification for a member product in the product line, rendered in terms of the specific features from the feature catalogue that are chosen for the product. In other words, it is a feature-based product specification, defined as an instantiation of the available feature options in the feature catalogue. A bill-of-features portfolio comprises the bill-of-features for each member product in a product line.

Bill-of-features portfolio engineering, described in 8.4, is the activity that produces and manages a bill-of-features for each member product in the product line.

6.2.4 Shared asset supersets

Shared assets are the digital artefacts associated with the software and systems engineering lifecycle of the product line. They are the digital building blocks for the member products in the product line. Shared assets can be whatever digital artefacts that compose a part of a delivered product or support the engineering process to create and maintain a member product.

The shared asset supersets illustrated in Figure 4 are organized into a systems engineering 'V', where each parallelogram in the 'V' represents a shared asset superset. For example, the parallelogram in the upper-left can be the system requirements shared asset superset, the parallelogram just below that can be the system design shared asset superset, and the three adjacent parallelograms just below that can be the mechanical design shared asset superset, the electrical design shared asset superset, and the software source code shared asset superset. Each parallelogram on the right-hand side of the 'V' is a verification and validation shared asset superset for its mirror image counterpart on the left-hand side of the 'V'.

Shared asset supersets include variation points, which are places in the asset that denote content that is configured according to a formal mapping from a bill-of-features to the form of content that meets the needs of that bill-of-features. Examples of possible types of configuration mappings include omission or selection of optional content; selection from among mutually exclusive content alternatives; generation of content from feature specifications; and feature-based transformation of content from one form into another.

In addition to variation points, shared asset supersets also contain content included in all member products. This content is not part of any variation point and is said to be common.

Shared asset superset engineering, described in 8.5, is the activity involved in the creation, development, evolution, and management of the product line's shared asset supersets.

6.2.5 PLE factory configurator

The PLE factory configurator is the mechanism that automatically produces product assets for a specific member product. In feature-based PLE, the configurator is an automated tool, as opposed to a manual process. It performs its task by processing the bill-of-features for that product and exercising the shared assets' variation points to produce configured content, in light of the feature selections made in that bill-of-features.

Use of the configurator, described in 8.6, provides the abstraction-driven automation that eliminates the activity of manually assembling and modifying engineering assets for each member product in the product line.

6.2.6 Product asset instances

Product asset instances are product-specific assets, automatically produced by the PLE factory configurator, derived from the shared asset supersets according to a bill-of-features for a specific member product.

The product asset instances illustrated in Figure 4 are organized into three systems engineering 'V's', where each 'V' represents a member product. Each parallelogram in a 'V' represents a product asset instance and corresponds to its isomorphic shared asset superset parallelogram in the shared asset supersets 'V'.

Product asset instances are in a form suitable for use in a particular member product and therefore should be suitable for testing and deployment in the context of a particular member product. Product verification, validation, and delivery of product asset instances, described in 8.7, is the process by which the configured assets are validated and delivered to the next lifecycle stage (e.g. manufacturing), to the customer, or to the market.

6.3 Relationships among the key elements of the factory

6.3.1 General

There are conceptual groupings among the elements of a PLE factory that provide insight into the overall feature-based PLE approach and the benefits that derive from the abstraction and automation. These are described in Figure 5, utilizing the tabular row headings and column headings.

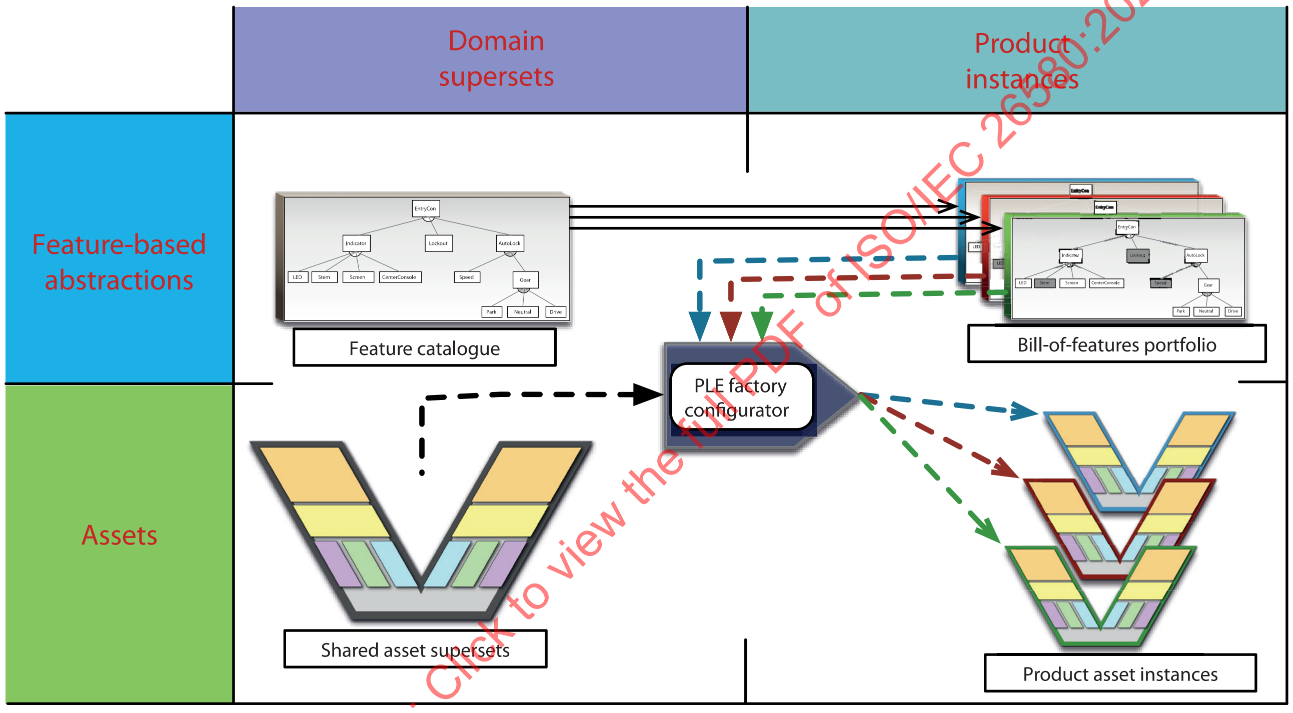

Figure 5 — Categories of relationships among key elements of the feature-based PLE factory

6.3.2 Feature-based abstractions: feature catalogue and bill-of-features portfolio

Feature-based abstractions refers to the top half of the PLE factory, as shown in the bold black rectangle in Figure 6. The feature-based abstractions contain the features relevant to the product line and its individual member products. Features, whether in a catalogue or a bill-of-features, are abstractions because they refer to capabilities and other distinguishing characteristics without referring to specific implementations or realizations.

There can be, by one to several orders of magnitude, fewer features in the feature catalogue than associated variation points in the assets, thereby reducing the cognitive burden of comprehending and managing a product line and the variations among the members of the product line.

Using a computer programming language analogy, the feature catalogue is a class or data type. Each member of the bill-of-features portfolio is an instance of the feature catalogue data type that represents all of the feature selections made for a specific member product.

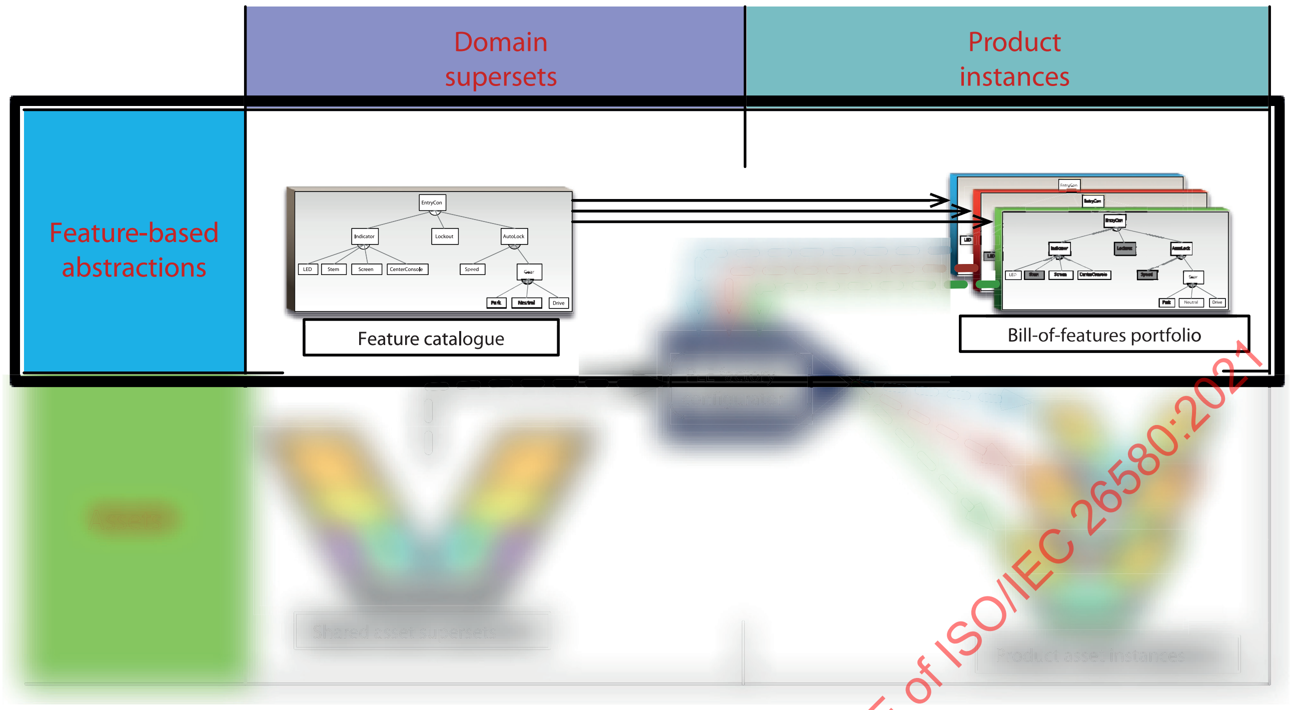

Figure 6 — Feature-based abstractions in the feature-based PLE factory

6.3.3 Domain supersets: feature catalogue and shared asset supersets

Domain supersets refers to the left-hand side of the PLE factory, as shown in the bold black rectangle in Figure 7. It comprises the feature catalogue and the shared asset supersets. Both PLE factory components are independent of any one specific member product, and instead have scope that extends over the entire product line.

The formal mapping between shared asset superset variation points and features in the feature catalogue provides the semantics of the abstraction layer and is sufficiently precise to enable the automation of the PLE factory configurator.

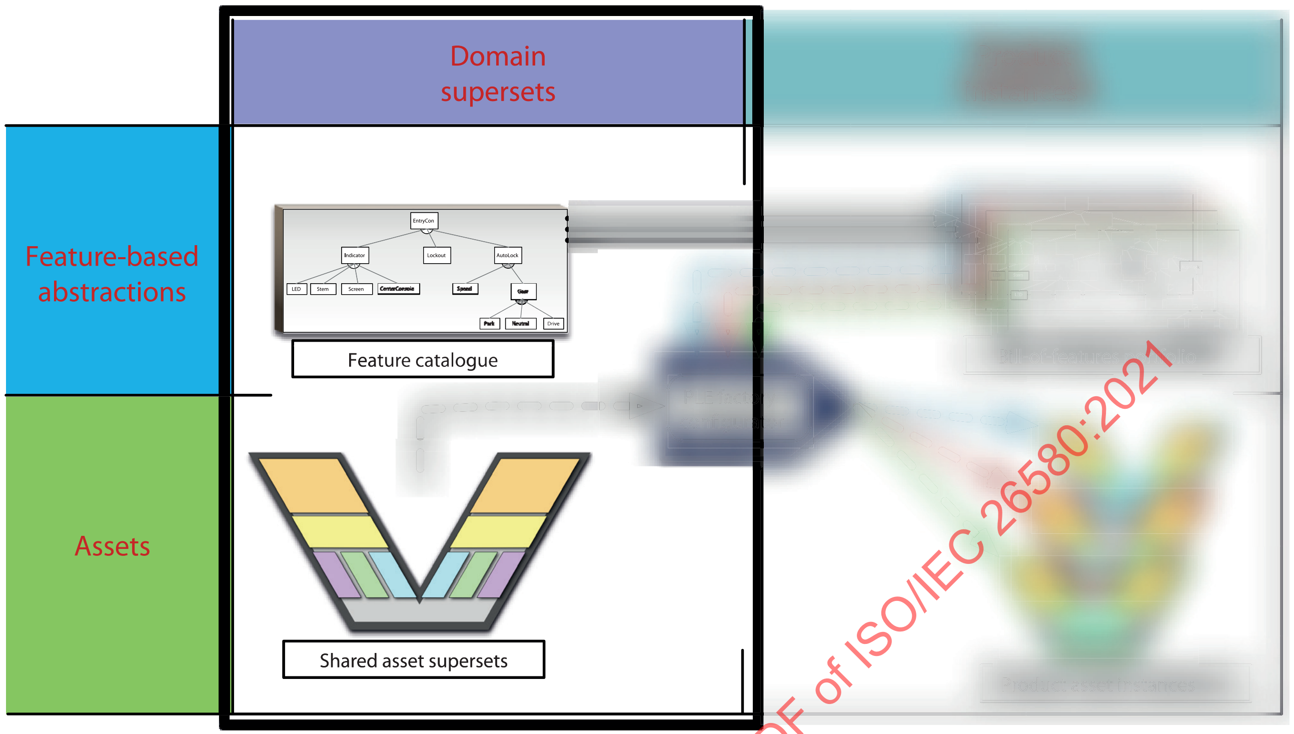

Figure 7 — Domain supersets in the feature-based PLE factory

6.3.4 Assets: shared asset supersets and product asset instances

Assets refer to the bottom half of the PLE factory, as shown in the bold black rectangle in Figure 8. They are the digital engineering assets involved in the engineering, deployment, and operations of the end products, including the shared asset supersets that support the entire product line and the product asset instances for each of the member products in the product line.

The PLE factory configurator provides the abstraction-driven automation to derive each product asset instance from the shared asset supersets by exercising the shared assets' variation points to configure content according to the feature selections made in a bill-of-features. This key benefit of the approach eliminates the labour-intensive and error-prone activity of manually assembling and modifying engineering assets for each member product.

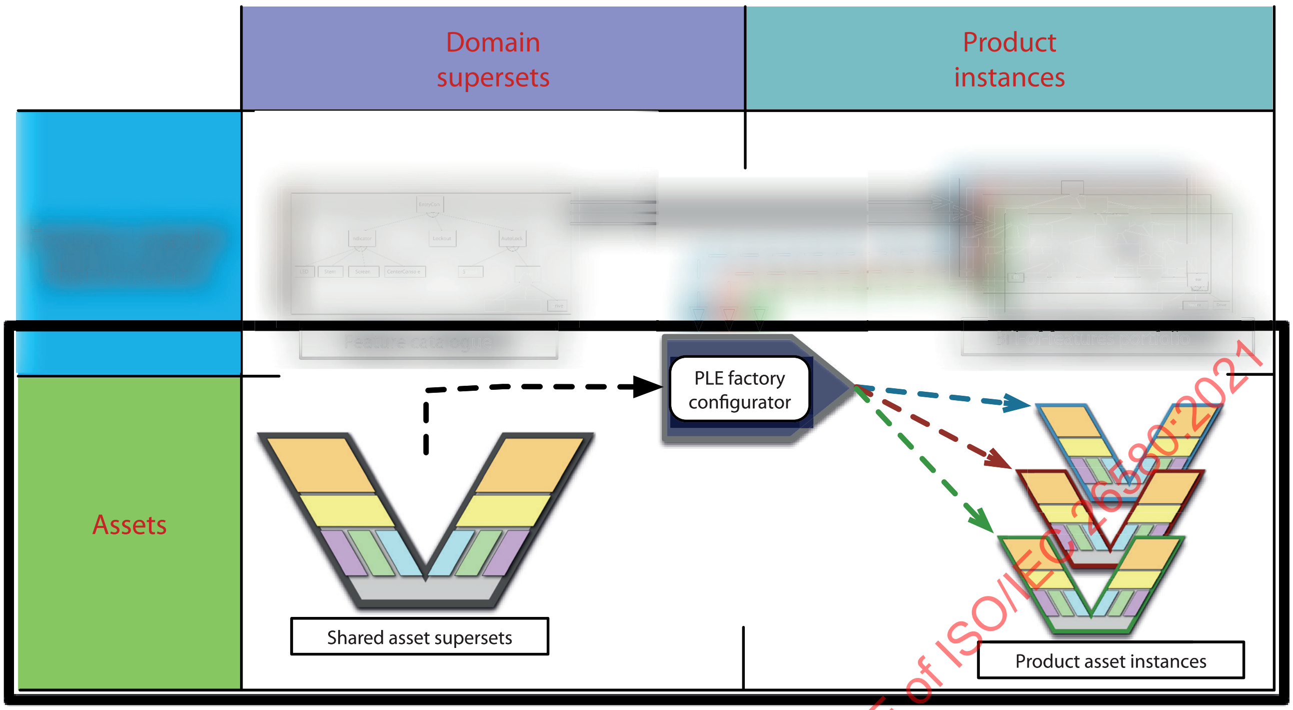

Figure 8 — Assets in the feature-based PLE factory

6.3.5 Product instances: bill-of-features portfolio and product asset instances

Product asset instances, by contrast to the shared asset supersets, are specific to an individual member product in the product line and are shown on the right-hand side of the PLE factory in the bold black rectangle in Figure 9. Each member of the bill-of-features portfolio contains a product-specific feature-based description, corresponding to one member of the product asset instances. Each member of the bill-of-features portfolio is an instance of the feature catalogue feature set; and each member of the product asset instances, emitted by the PLE factory configurator, is a product-specific instance of the shared asset supersets.

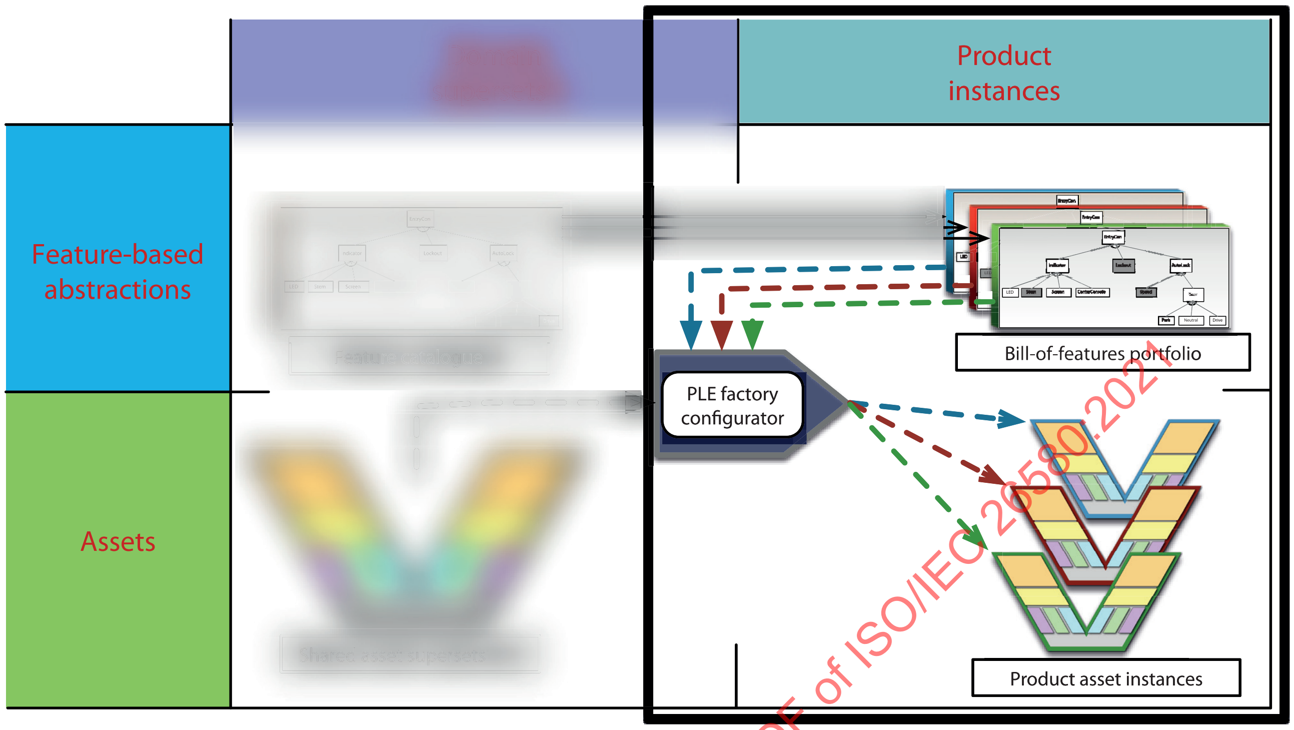

Figure 9 — Product instances for member products in the feature-based PLE factory

6.4 Reference model layers

Completing the feature-based PLE reference model are three layers, shown along the top of Figure 3. The layers characterize what stakeholders do in order to create, utilize, and sustain the feature-based PLE factory approach:

— The base technology layer comprises the tools and technology of the factory infrastructure. Think of this as the fully functional factory without any of the people inside to run the factory.

— The middle technical organization management layer focuses on the people, roles, and processes that operate a PLE factory. In combination, the base technology layer and the middle technical organization management layer provide a fully operational feature-based PLE factory capable of producing the assets for all of the products in a product line.

— The top business organization management layer focuses on the people, roles, and processes that utilize and leverage a PLE factory to achieve the business objectives of the enterprise. Using the analogy to a conventional factory, the top business organization management layer provides guidance and support for the executive leadership working in the office high-rise that overlooks the factory.

The distinct separation of stakeholder concerns in this document, between the feature-based PLE technology providers and the feature-based PLE technology users, is clearly visible in these layers. Stakeholders in the technology layer are tool providers and implementors concerned with what is required to create commercial off-the-shelf tools and technology to support the feature-based PLE practitioners in the other two layers. Stakeholders in the technical organization management layer and business organization management layer are practitioners, managers, and business leaders concerned with the methods required to effectively use commercial off-the-shelf tools and technology to achieve their technical and business objectives for feature-based PLE.

Together, these three layers of the feature-based PLE reference model provide guidance on what is required to achieve the benefits of feature-based software and systems product line engineering. Each layer is described in detail in Clauses 7 to 9 respectively.

6.5 Feature language

To bring this all together, the PLE factory shall have a well-defined feature language; that is, a syntax and semantics for the formal representation, structural taxonomy, and the relationships and mappings among the concepts and constructs in the feature catalogue, bill-of-features portfolio, and shared asset superset variation points. Figure 10 denotes the areas of the technology layer that are concerns for the feature language, which is elaborated along with examples in 7.2.

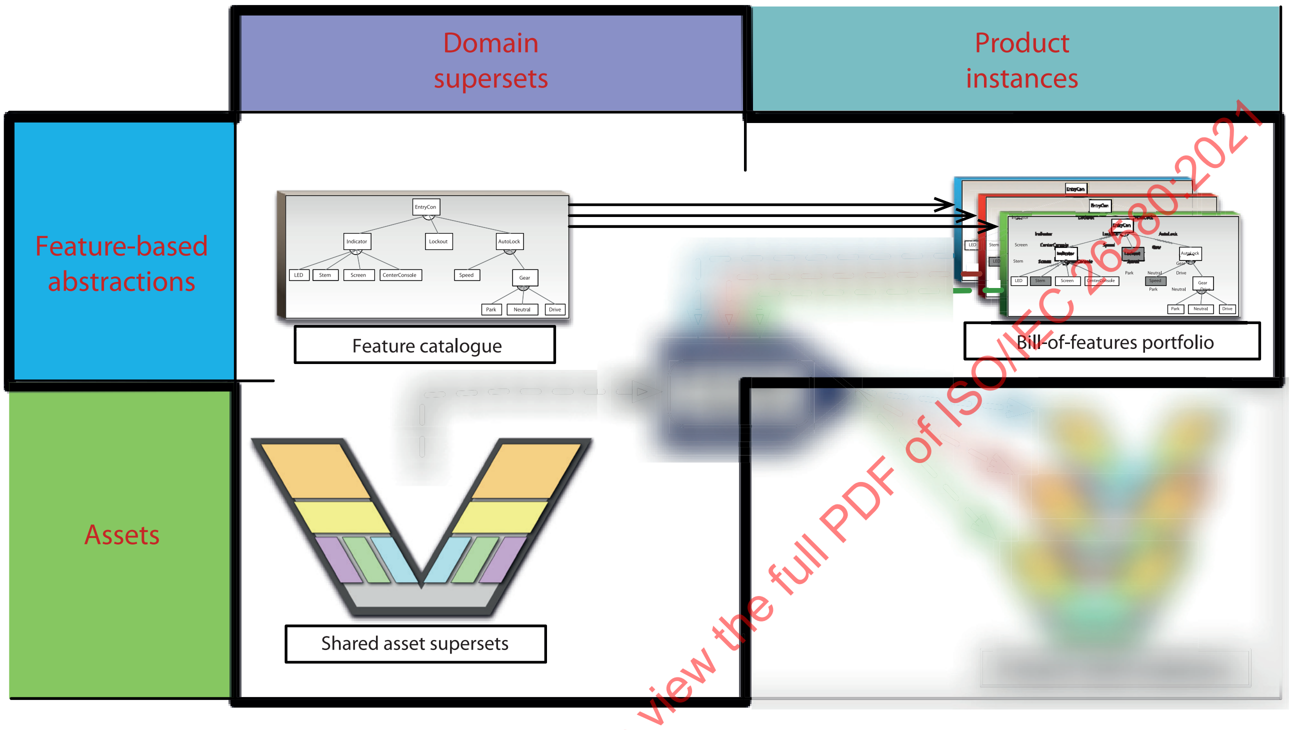

Figure 10 — Feature language relating all feature-based concepts in the PLE factory

6.6 Support for a hierarchical product line of product lines

Feature-based PLE should be applied hierarchically to produce a product line of product lines. Any product line can be decomposed into a collection of smaller product lines that are then hierarchically structured as child product lines. This decomposition can be applied recursively to any depth to create a tree-structured product line of product lines. This product line hierarchy enables mirroring of the recursive decomposition and structuring of a product architecture as a system-of-systems or system-of-subsystems and allows cooperative PLE factory development across organizational boundaries.

For example, a product line of automotive vehicles is composed of a product line of engines, a product line of brake systems, a product line of infotainment systems, a product line of interior lighting systems, and many more. Each vehicle in the vehicle product line comprises a selection of an engine, a selection of a brake system, a selection of an infotainment system, a selection of an interior lighting system, and so on.

Each product line in such a hierarchy has its own PLE factory. The member products of a product line constitute a set of choices from which member products in the parent product line can choose.

It is a further requirement of the overall feature language to support this flow of choices from child to parent, and to allow sharing of features among product lines by establishing an import relationship. The latter is crucial for establishing feature constraints and asset variation points among interrelated subsystems (e.g. a high-end cruise control that slows the car if there's traffic ahead requires a braking system that supports braking via software command).

In such a case, each constituent product line is turned out by its own PLE factory, and the overall product line (of product lines) is produced by a PLE factory (comprising subsidiary PLE factories). In this document, where we refer to a PLE factory and where the product line is a product line of product lines, the reference applies equally to a constituent PLE factory or the highest-level PLE factory.

6.7 Other concerns

6.7.1 General

Other conventional engineering disciplines are relevant to the practice of feature-based PLE. However, they are separate concerns and not specific to or limited to feature-based PLE, so are not explicitly represented in Figure 3 as key elements of this reference model. 6.7.2 to 6.7.5 summarize the ways in which these separate disciplines interplay with the feature-based PLE tools and methods.

6.7.2 Configuration management concern

The feature catalogue, shared asset supersets, and bill-of-features portfolio all evolve over time. The configuration management concern involves the temporal management – versions, branches, baselines, effectivity, check-in/check-out – of these technical elements in a PLE factory. This concern is addressed by the configuration management process (8.8).

Product asset instances are generated, so they can be treated as transient digital data. Because they can be discarded and regenerated at a later time as needed, they are not strongly implicated in the configuration management concern during development. However, product asset instances can elevate into temporal management as products transition from development into deployment and downstream operations such certification, creating contractual delivery archives, populating manufacturing bill-of-materials, creating digital twins, and so forth.

Modification and extension of the generated product asset instances and the optional merging of those changes back into the shared asset supersets is not included as a practice in this document, although if business processes dictate, it may occur and should follow the application engineering practices defined in the more general ISO/IEC 26550.

To support the configuration management concern of feature-based PLE, a PLE factory shall provide a capability to manage evolution over time to support versions, branches, baselines, effectivity, and check-in/check-out of the constituent parts of the feature catalogue, the bill-of-features portfolio, the shared asset supersets, and, if needed as described in the previous two paragraphs, product asset instances.

Temporal management may be built into the tools that, respectively, are used to create and maintain these elements, or it may be provided by separate tooling such as a configuration management system.

6.7.3 Traceability concern

Feature-based PLE supports the full set of assets in the software and systems engineering lifecycle 'V', including all of the relationships and traceability across the assets in the different stages such as requirements, design, implementation, verification, and validation. Beyond conventional traceability among assets, the traceability concern also includes the mapping relationship between the feature catalogue and the feature-based variation points in all of the shared asset supersets. This concern is addressed by the PLE factory configurator (7.7) and the traceability management process (8.9).

To support the traceability concern of factory-based PLE, a PLE factory shall include tooling to:

— create, modify, delete, and browse traceability relationships among elements in the feature catalogue and shared asset supersets.

To support the traceability concern of factory-based PLE, a PLE factory should include tooling to:

— transform traceability links in the shared asset supersets into the corresponding traceability links in the product asset instances after a configuration operation by the PLE factory configurator, supporting a well-defined semantics for transforming traceability links that go into and out of variation points that are transformed by the configurator; for example, when a superset traceability link exists between an optional requirement variation point and an optional test case variation point, if during product configuration both of these variation points are omitted, then the PLE factory tooling should also remove the extraneous traceability link;

— analyse and report on traceability links in the product asset instances after a configuration operation by the PLE factory configurator; for example, after a product configuration operation, if either the source or destination of a traceability link does not exist due to omitted content, and if this condition is considered an error, then the PLE factory tooling should report this "broken" link.

Traceability capabilities may be provided by the PLE factory development environment or may be provided by separate traceability tooling.

6.7.4 Change management concern

A PLE factory is subject to change requests from customer, business, and technical stakeholders. To assure that the PLE factory evolves to best meet the overall needs of all the stakeholders, change requests shall be reviewed, adjudicated, prioritized, allocated to temporal management baselines, and decomposed for implementation in the feature catalogue, bills-of-features, and shared asset supersets. Because product asset instances are generated, they are not implicated in the primary change management concern for a PLE factory. This concern is addressed by the change management process (8.10).

Change management may be built into the tools that, respectively, are used to create and maintain the PLE factory elements, or it may be provided by separate change management tooling.

6.7.5 Access control concern

Intellectual property and security concerns require access control to shared asset supersets, feature catalogue, bill-of-features, and product asset instances.

To support the access control concern, a PLE factory should include tooling to enforce role- or identity-based access controls to content.

7 Technology layer

7.1 General

This clause describes the tool and technology elements necessary to support the aspects of feature-based PLE shown in the technology layer of the reference model in Figure 3. It is therefore the concern of the feature-based PLE technology providers who want to offer commercial off-the-shelf tools.

The technology layer and this document as a whole are independent of the data and behavioural implementation methods, models, languages, tools, and techniques used to realize and practice feature-based PLE. Implementation choices are intentionally left open within black box descriptions in this clause, so that commercial off-the-shelf tool and technology providers have the flexibility to make their own implementation choices. For the key elements of the PLE factory, examples are provided that may be used for their black box implementations.

7.2 Feature language

To support feature catalogue engineering, bill-of-features portfolio engineering, and shared asset superset engineering, the technology layer of a PLE factory shall provide tooling that supports a well-defined feature language that provides the syntax and semantics for the formal representation, structural taxonomy, and relationships among the concepts and constructs in the feature catalogue, bill-of-features portfolio, and shared asset superset variation points, including:

— A data declaration language for representing a feature catalogue – the abstractions for the differentiating characteristics in any product line. The feature catalogue is a collection of choices that are offered, which may be expressed in the data declaration language with any of the following or others:

- simple optional choice, specifying that a choice is to be made to include or exclude a feature;
- mutually exclusive choice, specifying that one and only one selection is to be made from a list of available choices for a feature;
- mutually inclusive choice, specifying that zero or more selections is to be made from a list of available choices for a feature;
- aggregate choice, specifying that all of a fixed list of available choices is to be made for a feature;
- hierarchical choice with sub-choices, specifying that any member in the list of available choices for a mutually exclusive, mutually inclusive, or aggregate feature can be another feature that requires further sub-choices to be made;
- scalar choice, specifying that a value is to be selected from a range of values, such as picking an integer value between 1 and 10.

— A feature constraint language in the feature catalogue to express relationships for valid combinations of feature choices in a bill-of-features. The relationships in the feature constraint language may be expressed with any of the following or others:

- mathematical relationships on scalar features; for example, if NumberOfDoors is an integer feature expressing the choice for the number of passenger doors on an automobile, a feature constraint indicating that number of doors must be greater than one can be of the form (NumberOfDoors > 1);
- relational programming logic (i.e. N-ary Boolean expressions comprising typed variables plus operators such as AND, OR, NOT, Greater-than, Equal-to, Less-than); for example, a feature constraint of the form ((NOT AdaptiveCruiseControl) OR (Brakes == Electronic)) can be used to express that the optional AdaptiveCuiseControl feature is not selected (adaptive cruise control is the capability that enables an automobile to slow down and speed up to keep a safe distance from the car in front of it) or that the Brakes feature choice selected is Electronic (electronic breaking allows software to initiate breaking to slow down an automobile);
- graphical relationships in a visual feature language; for example, two selectable nodes in a visual graph with a symmetric arc between them labelled “Excludes”, indicating that the two nodes are mutually exclusive (i.e. one or the other can be selected, but not both).

— A data instantiation language for representing the bill-of-features selections from the feature catalogue. The bill-of-features is the collection of selections made for each of the choices offered in feature catalogue. The data instantiation language for the bill-of-features shall provide the means to assign values to choices for all of the constructs in the data declaration language of the feature catalogue.

— A variation point language to express asset variation point structures and to express the mapping between feature selections made in a bill-of-features and the automated variation point configurations. The variation point language may be expressed with any of the following or others:

- a means to define the extent of a variation point; for example, a variation point in requirements can be applied to a single requirement or to a section of requirements;
- inclusion or omission of optional content in a shared asset superset;
- selecting a variant of content in a shared asset superset;
- relational programming logic;

  - graphical relationships between the data declaration language of the feature catalogue and artefacts in a type of shared asset superset, such as graphical relationships between features in a feature catalogue and model elements in a UML model;
  - generation of content from a feature specification, at the location of a variation point in a shared asset superset; for example, a code generator can examine feature selections in a bill-of-features to generate a method call with parameter values;
  - feature-based transformation of shared asset superset content from one form into another, at the location of a variation point in a shared asset superset.
- A hierarchical decomposition structure for creating a product line of product lines.
- A construct to capture and represent auxiliary information such as comments and attributes for each object in the feature language, including at least information rendered in plain text.

For example, an overall feature language formalized in a tool from a PLE technology provider can be defined as:

- A feature catalogue with a flat list of features, where each feature is defined with a name and a mutually exclusive list of choices. PLE practitioners using the tool with this feature language would implement the feature catalogue for their PLE factory as a list of features for their product line, such as the following examples from the automotive domain:
  - Feature Name: AdaptiveCruiseControl. Choices: Yes, No;
  - Feature Name: Braking. Choices: Basic, Electronic, Performance.
- Binary feature constraint expressions in the feature catalogue. PLE practitioners using the tool would implement binary feature constraints in their feature catalogue to assure valid feature selections, such as the following from the automotive example:
  - (AdaptiveCruiseControl == Yes) REQUIRES ((Braking == Electronic) OR (Braking == Performance)).
- A bill-of-features portfolio, where each bill-of-features has a name and comprises a selection for each of the features in the feature catalogue. PLE practitioners would use the tool to make selections in the bill-of-features for their portfolio, such as the following from the automotive example:
  - Bill-of-features Name: Sport;
    - Feature AdaptiveCruiseControl: Yes;
    - Feature Braking: Performance;
  - Bill-of-features Name: Entry;
    - Feature AdaptiveCruiseControl: No;
    - Feature Braking: Basic.
- A variation point definition for optional requirements, where the variation point extent is a single requirement and the variation point mapping is a traceability link between a feature option and an optional requirement variation point, indicating that the selection of the feature option in a bill-of-features will cause the optional requirement to be present in the corresponding product asset instance. PLE practitioners would use the tool to create variation points in their requirements shared asset superset, such as the following from the automotive example:
  - Feature choice Braking:Performance in the feature catalogue traces to optional requirement "High performance brakes shall support emergency braking from 180 miles per hour to full stop."
- A variation point definition for optional source code files, where the variation point extent is a single source code file and the variation point mapping is a traceability link between a feature option and

an optional source code file variation point, indicating that the selection of the feature option in a bill-of-features will cause the optional source code file to be present in the corresponding product asset instance. PLE practitioners would use the tool to create variation points in their source code shared asset superset, such as the following from the automotive example:

— Feature choice AdaptiveCruiseControl:Yes in the feature catalogue traces to optional source code file Front_Distance_Sensor_Controller.java.

7.3 Feature catalogue

To support feature catalogue engineering, the technology layer of a PLE factory shall provide tooling that includes:

— an editor with a user interface to view, create, modify, delete, and otherwise edit the feature models and other constructs and relationships in the data declaration language of the feature catalogue, as well as the feature constraint language for expressing relationships among feature choices offered;

— an operation to determine the semantic and syntactic correctness of the feature catalogue;

— a digital representation for all the constituent parts of the feature catalogue that can be persistently stored, read and modified on a computer storage system.

7.4 Bill-of-features portfolio

To support bill-of-features portfolio engineering, the technology layer of a PLE factory shall provide tooling that includes:

— a mechanism to create, name, rename, organize, and delete a bill-of-features as a member of the bill-of-features portfolio;

— an editor with a user interface to view, create, modify, delete and otherwise edit the selections made in the data instantiation language for a bill-of-features; the editor shall understand and respect the relationship between the feature catalogue as a data type and the bill-of-features as an instance of the selections made from the choices offered in the feature catalogue;

— an operation to detect and require that all of the feature constraints declared as part of the feature catalogue are satisfied in each bill-of-features member of the bill-of-features portfolio;

— a mechanism to either recommend or automatically make additional feature selections in a bill-of-features, based on previous feature selections made in that bill-of-features, constraints, and inferences;

— an operation to detect and maintain consistency between the feature catalogue and the bill-of-features portfolio; when the feature catalogue data type declaration changes, each bill-of-features member of the bill-of-features portfolio is resolved to accommodate the changes; for example, if an optional feature in the feature catalogue needs to evolve into a feature that offers three mutually exclusive alternatives, then existing instantiations of that optional feature in all of the bill-of-features need to be migrated to instantiations of that feature with the appropriate alternative selected;

— a digital representation for all the constituent parts of the bill-of-features portfolio that can be persistently stored, read and modified on a computer storage system.

7.5 Shared asset supersets

To support shared asset superset engineering, the technology layer of a PLE factory shall provide tooling that:

— allows attachment of an open-ended set of shared asset supersets to a product line so that the PLE factory configurator can configure them based on a bill-of-features for a member product to produce corresponding product asset instances;

— provides the mechanism to create and delete variation points in each asset type within a shared asset superset by engineers using the engineering tool they are accustomed to using to manipulate that kind of engineering asset; these should include requirements tools, system design tools, software development tools, electrical design tools, mechanical design tools, documentation tools, verification and validation tools, and other engineering tools to be used in a PLE factory;

— provides a consistent way to specify variation points using a single variation point language for every type of shared asset, as opposed to each type of shared asset having its own language of variation points, to aid in comprehension across the lifecycle stages and to assure that all shared assets can be systematically processed by the PLE factory configurator;

— provides an editor to view, create, modify, delete and otherwise edit the constituent variation point language constructs (as defined in 7.2) for all variation points, including the formal mapping from bill-of-features selections to the configured variation point content realization;

— provides a well-defined semantics for each asset type on how the PLE factory configurator will exercise variation points based on feature choices in a bill-of-features.

7.6 Product asset instances

The technology layer of a PLE factory shall provide tooling that allows inspection, verification, validation, building, packaging, archiving, and deployment, for transitioning products from development into operations, for all asset types in the product asset instances, using the engineering tools that engineers are accustomed to using for each type of asset.

7.7 PLE factory configurator

The technology layer of a PLE factory shall include a PLE factory configurator that provides the mechanism to configure the shared asset supersets attached to a product line into the product asset instances. Configuration occurs by exercising each variation point in each shared asset superset according to the feature choices in the bill-of-features for a product.

The PLE factory configurator shall also interact with the traceability tooling in the PLE factory to transform traceability links in the shared asset supersets into the semantically equivalent traceability links in the product asset instances. For example, superset traceability links between common artefacts should be replicated as instance traceability links in the corresponding common artefacts in the asset instances. Similarly, superset traceability links to or from variation points should be transformed in the asset instances based on the syntax and semantics of the variation point language.

7.8 PLE factory development environment

The technology layer of a PLE factory shall provide tooling that includes a PLE factory development environment. This should include:

— support for multiple, concurrent users;

— editors for browsing, creating, modifying, and navigating the product line hierarchy and its infrastructure;

— tools to aid in product line analysis and comprehension, such as:

  — syntactic and semantic checker for all elements of a product line;

  — search tool over a product line;

  — impact analysis tools, such as a tool to determine in which products a given feature or variation point is used;

  - analytics and statistics report on the product line, including coverage statistics for product line scope, ratios of common artefacts versus variation point artefacts, and so forth;
- a command line interface for integration into automated build systems and other tools in a development and production environment;
- an open programming interface so that third parties can develop their own integrations with the solution.

The PLE factory development environment should have an architecture and user interface that scales for the number of engineers involved in the engineering of the product lines and their varying experience with the various engineering tools and the product line engineering infrastructure.

8 Technical organization management layer

8.1 General

This clause describes the people, roles, and processes that operate a PLE factory, as shown in the technical organization management layer of the reference model in Figure 3. It is therefore the concern of the feature-based PLE engineering practitioners who use tools described in Clause 7 for the technology layer to achieve their technical objectives for feature-based PLE. Because of this distinct separation of concerns between technology users and technology providers, the tool and technology usage discussions in this clause refer back to the tool capabilities from the providers in Clause 7.

The technical organization management layer in conjunction with the technology layer is sufficient to describe a fully operational feature-based PLE factory that produces the product asset instances for all of the member products in a product line. Specifically, the technical organization management layer includes all of the processes and operations necessary to create, maintain, and utilize the key elements of a PLE factory as shown in Figure 4.

The reference model and this document as a whole are independent of the engineering and management processes used to create and manage the content of the shared asset supersets, other than their variation points. They are also independent of the engineering processes used for product verification and validation of the product asset instances. As such, the reference model is completely compatible with the process groups of ISO/IEC/IEEE 12207 and ISO/IEC/IEEE 15288. The technical organization management process is in line with the organizational project-enabling processes group and the agreement processes group of ISO/IEC/IEEE 12207 and ISO/IEC/IEEE 15288.

Similarly, the reference model and this document as a whole do not prescribe the use of any particular software and systems engineering methodologies. Organizations can decide which methodologies are appropriate for their needs.

Each operation is described in terms of the following attributes:

- the title of the operation;
- the purpose of the operation;
- the outcomes of the operation;
- the inputs to produce the outcomes;
- the tasks to achieve the outcomes and methods to carry out the tasks;
- the tools to support the tasks and methods.

Required tool support is in the purview of the technology layer, described in Clause 7.

8.2 Relationship to ISO/IEC 26550 technical management process group and ISO/IEC 26556

This document incorporates by reference the processes listed below, which are described in ISO/IEC 26550:2015, 6.1.1 (Domain engineering life cycle, product line scoping), in the roles

— product scoping;

— domain scoping;

— asset scoping.

This document incorporates by reference the processes listed below, which are described in ISO/IEC 26550:2015, 6.4.4 (technical management process group, support management), in the roles

— technical quality management;

— decision management;

— technical risk management;

— tool management.

This document also incorporates by reference the processes listed below, which are described in ISO/IEC 26556:2018, Clauses 5 to 7:

— organizational-level product line planning (ISO/IEC 26556:2018, Clause 5):

  — customer relationship management (ISO/IEC 26556:2018, 5.3);

  — developing a sourcing strategy (ISO/IEC 26556:2018, 5.4);

  — organizational deployment and innovation planning (ISO/IEC 26556:2018, 5.5);

  — organizational operations planning (ISO/IEC 26556:2018, 5.6);

— organizational product line enabling (ISO/IEC 26556:2018, Clause 6):

  — structuring the product line organization (ISO/IEC 26556:2018, 6.2);

  — organizational product line infrastructure (ISO/IEC 26556:2018, 6.3);

  — organizational product line quality management (ISO/IEC 26556:2018, 6.4);

  — organizational governance through product family management (ISO/IEC 26556:2018, 6.5);

— organizational product line managing (ISO/IEC 26556:2018, Clause 7):

  — product line deployment and innovation management (ISO/IEC 26556:2018, 7.2);

  — operations management (ISO/IEC 26556:2018, 7.3);

  — organization-level product line monitoring and control (ISO/IEC 26556:2018, 7.4);

  — organizational product line risk management (ISO/IEC 26556:2018, 7.5);

  — product line evolution management (ISO/IEC 26556:2018, 7.6).

The following provisos apply to the incorporated material:

a) References to “domain engineering” should be replaced by “feature catalogue engineering and shared asset superset engineering” (see 6.3.3 and Figure 7).

b) References to “application engineering” should be replaced by “bill-of-features portfolio engineering and product verification, validation, and delivery” (see 6.3.5 and Figure 9).

8.3 Feature catalogue engineering

8.3.1 Purpose

Feature catalogue engineering is the activity that produces and maintains a feature catalogue that captures all of the germane distinguishing characteristics among the products in the product line – that is, distinguishing characteristics that will lead to differences in the product asset instances. The variability is based on things like regional variation, low-end versus high-end offerings, consumer options, non-functional performance variations, and so forth.

8.3.2 Role

— Feature catalogue engineer.

8.3.3 Outcomes

The outcome of feature catalogue engineering is a new or updated feature catalogue for a product line, expressed in the feature language of the feature catalogue, as described in 7.2.

The feature catalogue includes captured constraints about allowable combinations of feature choices.

A feature catalogue typically comprises several feature models. In this way, the feature catalogue can be created (as well as shared and comprehended) in small, intellectually manageable and maintainable pieces, reflecting a decomposition structure that results from a separation-of-concerns approach to allocating features to feature models.

8.3.4 Inputs

— Artefacts currently in use to capture and manage feature variations. This may include documents, spreadsheets, models, build scripts, and so forth that describe optional and varying capabilities or characteristics of the products in the product line.

— Summary of the ways in which the products in the product line will, and/or currently do, differ from each other. This summation may be in the form of expert knowledge on the part of those responsible for those products, or a comparison of requirements for the different products.

— Characterization of the different products that are currently (or planned to be) built for delivery by the PLE factory to which the feature catalogue belongs.

— A change request to create, extend, or refine the feature catalogue.

— An empty or preexisting feature catalogue that will be extended or otherwise refined by the feature catalogue engineering operation.

8.3.5 Tasks

8.3.5.1 Design and review decomposition of overall feature catalogue

Almost always, a single, monolithic feature catalogue is far too large and unwieldy to be practical. Instead, the feature catalogue is decomposed (using some good separation of concerns criterion) into a manageable number of separate feature models. The output of this work is a collection of feature models and an informal description of the purview of each one.

Methods should support:

— application of separation of concerns and encapsulation;

— facilitated elicitation to capture information from domain experts;

— review of the outcome;

— use of the tool capabilities described in 8.3.6 to express the collection of feature models identified in this task, according to the representation provided by the feature language in the tool.

8.3.5.2 Design and review a feature model's features

This work involves defining the features that a feature model (identified through the work of the task described above) will comprise, and (where needed for clarify, training, or domain information capture) descriptive commentary for each. This work also involves defining the name for each feature, in accordance with sound naming conventions, as well as providing (where needed) descriptive commentary to explain the feature.

This task typically involves an elicitation and capture activity from individuals possessing knowledge about the domain involved, and/or the systems that constitute the product line. The elicitation activity can reflect the decomposition of the feature catalogue into constituent feature models; that is, a particular elicitation activity may be limited to one or a few specific feature models.

Methods should support:

— facilitated elicitation to capture information from domain experts;

— document analysis and review to identify existing descriptions of variation;

— difference analysis to identify places where existing instances differ (for example, code files built for two separate member products);

— review of the resulting feature model;

— use of the tool capabilities described in 8.3.6 to express the features in the feature model identified in this task, according to the representation provided by the feature language of the tool.

8.3.5.3 Design and review a feature model's feature constraints

The goal of this task is to identify any dependencies and relationships between the different features of the product family. These constraints may be discovered and reverse engineered from requirements with statements like, "if Feature *X* or Feature *Y* is included, then Feature *Z* is not allowed." This task also includes capturing domain- or application-specific descriptive commentary that explains why the constraints are true.

Methods should support:

— facilitated elicitation to capture information from domain experts;

— review of the resulting feature constraints;

— (once the constraints are captured in the tooling) validation to test that the tooling will detect and report constraint violations;

— use of the tool capabilities described in 8.3.6 to express the feature constraints identified in this task, according to the representation provided by the feature language of the tool.

8.3.6 Tools

Tooling for all tasks is as described in 7.8, which describes the base PLE factory development environment capabilities, 7.2, which describes the feature language in which to express the feature catalogue, and 7.3, which describes the tooling necessary for users to view create, modify, delete, and otherwise edit the feature catalogue.

For example, an editor for a feature catalogue based on a flat list of features with mutually exclusive options can provide feature catalogue engineers with a simple table-style user interface, with rows and columns for feature names and the names for available feature options, whereas an editor for a feature catalogue based on a multilevel feature tree can provide a drag-and-drop style graphical editor user interface for the nodes and branches in the feature tree.

8.4 Bill-of-features portfolio engineering

8.4.1 Purpose

Bill-of-features portfolio engineering is concerned with the creation and maintenance of a feature-based specification (a bill-of-features) for each member product.

8.4.2 Role

— Bill-of-features portfolio engineer.

8.4.3 Outcomes

The primary outcome of this activity is a bill-of-features for each member product.

A change request for the feature catalogue sometimes results from this activity, in the case where a feature needed to specify the product in a way that distinguishes it from other products was not present in the feature catalogue.

8.4.4 Inputs

— The feature catalogue, which contains the features (and constraints among them) that can be selected to form a bill-of-features.

— Any information about the product whose bill-of-features is being specified. This may include information from the business part of the organization, a portfolio roadmap, requirements specification, a marketing report, a customer order, a user manual, an architectural or design specification, or experience using or operating the product.

— A change request to create, extend, or refine one or more bill-of-features.

— An empty or preexisting bill-of-features portfolio that will be extended or otherwise refined by the bill-of-features portfolio engineering operation.

8.4.5 Tasks

8.4.5.1 Create and review bill-of-features for a member product

The primary goal of this activity is to produce a bill-of-features for one member product. This task should be repeated for each member product, to produce the bills-of-features for all member products.

Methods should support:

- facilitated elicitation to capture information from product portfolio experts about desired member products in the portfolio;
- facilitated elicitation to capture information from product experts about desired characteristics for each member product in the portfolio;
- reviewing the selections to ensure correctness and consistency with the needs of the product line;
- creating a change request, if necessary, to extend or refine the feature catalogue;
- use of the tool capabilities described in 8.4.6 to
  - create, name, and store a bill-of-features;
  - fill out the specification for a bill-of-features by making selections from the choices offered by the feature catalogue;
  - update a bill-of-features in response to changes in need of the corresponding member product, or changes in the feature catalogue;
  - add descriptive information, as appropriate, to the bill-of-features.

8.4.5.2 Check the selections in a bill-of-features for consistency with all of the constraints captured in the feature catalogue

The primary goal of this activity is to ensure that no errors have been made in the selections.

Methods should support:

- use of the tool capabilities described in 8.4.6 to check for errors in the bill-of-features selections;
- understanding the source of any errors found;
- remediating the errors, either by modifying the selections made or by changing the constraints expressed in the feature catalogue via a change request.

8.4.6 Tools

Tooling for all tasks is as described in 7.8, which describes the base PLE factory development environment capabilities and 7.4, which describes the tooling for users to create, name, organize, define, verify, and manage a bill-of-features as a member of the bill-of-features portfolio.

For example, a bill-of-features editor corresponding to a feature catalogue editor with a table-style user interface and a flat list of features with mutually exclusive options can provide bill-of-features engineers an extended feature catalogue user interface with a column next to the feature options for indicating which options are selected. Similarly, a bill-of-features editor corresponding to a feature catalogue editor with a multilevel feature tree can provide bill-of-features engineers an extended feature catalogue user interface with a point-and-click checkbox in each feature option node to indicate which options are selected.

8.5 Shared asset superset engineering

8.5.1 Purpose

Shared assets are the “soft” digital assets associated with the engineering lifecycle of the products, the building blocks of the products in the product line. Shared assets shall include whatever assets are representable digitally and either compose a product or support the engineering process to create a product.

8.5.2 Role

— Shared asset superset engineer. There will be shared asset superset engineers for each type of asset managed by the PLE factory, such as requirements superset engineers, design model superset engineers, source code superset engineers, test case superset engineers, user documentation superset engineers, and so forth.

8.5.3 Outcomes

The primary outcome of this process is, of course, the shared asset supersets with variation points that are available to be configured by the PLE factory configurator to produce product asset instances for individual member products.

Another outcome can be change requests for updates to the feature catalogue. It can be the case that the feature catalogue is missing a feature with which to express a necessary variation point in a shared asset superset. In this case, the shared asset superset engineer will inform the change control authority for the feature catalogue of the need. The outcome can be a new feature or an updated feature; or, it can be the case that the variation can be expressed using a feature or combination of features already in the feature catalogue.

8.5.4 Inputs

— Inputs to shared asset superset engineering may include the assets associated with a particular product or set of products, in the case where a product line is being formed through the merger of already-existing products. In this case, the individual projects' assets can be merged, factoring out common material and eliminating duplication.

— The feature catalogue, which contains the features (and constraints among them) that can be used to define the formal mapping from bill-of-features selections to the configured variation point content realization.

— A change request to create, extend, or refine one or more shared assets.

— An empty or preexisting shared asset superset that will be extended or otherwise refined by the shared asset superset engineering operation.

8.5.5 Tasks

8.5.5.1 Develop shared asset superset

The goal of this task is to create, extend, or otherwise refine a shared asset superset and attach it to a PLE factory so that the PLE factory configurator can configure it to produce product asset instances for a product family.

Methods should support:

— merging of preexisting assets (such as those associated with preexisting products that are going to be brought under the feature-based PLE umbrella);

— identifying commonalities and differences among the preexisting assets;

— demarcating the content in the superset corresponding to differences;

— use of tooling as appropriate for the asset type, which is not otherwise constrained by the PLE factory tools described in 8.5.6 for this task.

8.5.5.2 Insert variation points in shared asset supersets

The goal of this task is to have a shared asset superset that contains variation points that the PLE factory configurator can use to correctly produce a product asset instance based on a bill-of-features.

Methods should support:

— identifying places in the shared asset superset where a variation point should be created;

— creating variation points;

— updating existing variation points;

— expressing the variation point configuration behaviour using the variation point vocabulary in the feature language, which may include inclusion or omission of the content, selection from among mutually exclusive content alternatives, generation of content from feature specifications, and feature-based transformation of content from one form into another;

— issuing a change request for the feature catalogue if it does not current provide the features to express the necessary variation;

— use of the tool capabilities described in 8.5.6 to express each variation point identified in this task, including the location and extent, as well as the formal mapping from bill-of-features selections to the configured variation point content realization, according to the representation provided by the feature language of the tool.

8.5.5.3 Test variation points

The goal of this task is to ensure that the variation points have been correctly specified.

Methods should support:

— analysis to determine the possible combinations of feature selections that affect this variation point;

— applying each combination of feature choices to a variation point and examining the result;

— ensuring that the result is correct;

— updating the variation point when a relevant feature changes as a result of a change in the feature catalogue;

— use of the tool capabilities described in 8.5.6 to individually exercise each variation point identified in this task, to verify the desired behaviour under different feature combinations.

8.5.6 Tools

Tooling for all tasks is as described in 7.8, which describes the base PLE factory development environment capabilities and 7.5, which describes the tooling necessary to let users create, validate, and maintain shared asset supersets and feature-based variation points within those shared asset supersets.

For example, a PLE-enabled requirements management tool can provide requirements superset engineers a point-and-click command to indicate that a requirement is a variation point, followed by the presentation of a popup graphical representation of the feature catalogue that would allow the engineer to click on a feature option in order to establish the mapping between the feature option and the optional requirement variation point.

8.6 Automated configuration of the product asset instances

8.6.1 Purpose

The purpose of this process is to invoke the PLE factory configurator to produce product asset instances for a member product.

8.6.2 Role

— PLE factory build engineer.

8.6.3 Outcomes

— Product asset instances configured correctly according to the bill-of-features provided.

8.6.4 Inputs

— Bill-of-features for a product;

— Shared asset supersets attached to a PLE factory.

8.6.5 Task — Configure the shared asset supersets using the PLE factory configurator

The goal of this task is to ensure that the output of a PLE factory satisfies its requirements and is ready for delivery.

Methods should support:

- utilising the PLE factory development environment to produce product asset instances corresponding to a bill-of-features;
- analysing any errors or anomalies reported by the PLE factory configurator;
- remediating any errors or anomalies reported by the PLE factory configurator;
- use of the tool capabilities described in 8.6.6 to select the bill-of-features identified in this task and then invoke the PLE factory configurator operation to produce product asset instances.

8.6.6 Tools

Tooling for all tasks is as described in 7.8, which describes the base PLE factory development environment capabilities and 7.7, which describes the configuration functionality of the PLE factory configurator.

For example, the PLE factory configurator user interface can provide the PLE factory build engineer with a point-and-click button to invoke the configurator, which would present a popup menu for selecting a bill-of-features from the bill-of-features portfolio and produce the product asset instances according to the selected bill-of-features.

8.7 Verification, validation, and product delivery of the product asset instances

8.7.1 Purpose

The purpose of this activity is to take the output of a PLE factory and perform all steps necessary in support of delivery.

Reviews of stand-alone content, such as unit testing for stand-alone software components, should be performed on the shared asset supersets and not be repeated for each product asset instance. The PLE organization should undertake a study to determine which other V&V (verification and validation) activities may be effectively carried out once on the shared asset supersets and not once per product asset instance, using criteria such as impact of defects, pedigree and historical reliability and stability of content, sensitivity to context, and other criteria as appropriate.

8.7.2 Role

— Product asset instance verification engineer.

— Product asset instance validation engineer.

— Product asset instance delivery engineer.

8.7.3 Outcomes

— Product asset instances verified and validated.

— Product asset instances packaged and bundled as appropriate for delivery.

— Any defects encountered reported to the PLE factory change control authority.

8.7.4 Inputs

— Product asset instances produced by the PLE factory configurator.

8.7.5 Tasks

8.7.5.1 Verify and validate member product

The goal of this task is to ensure that the output of the PLE factory satisfies its requirements and is ready for delivery. By "requirements" we include any property the member product must have in order to be deliverable. This includes functional correctness, quality attribute achievement, regulatory and process compliance, and whatever other properties may apply.

Methods are as appropriate for the member products and are not otherwise constrained by this document.

8.7.5.2 Package and deliver a member product

The goal of this task is to perform any necessary post-processing on the output of the PLE factory and prepare the member product for delivery, ensuring that any required deliverables are included in the package.

Methods are as appropriate for the member products and are not otherwise constrained by this document.

8.7.5.3 Identify and report a defect

The goal of this task is to ensure that defects are reported to the change control authority for the PLE factory, so that changes to eliminate the defect are carried out inside the PLE factory to the feature catalogue, the bill-of-features for the product, and/or the shared asset supersets, as appropriate.

Methods are as appropriate for the member products and are not otherwise constrained by this document.

8.7.6 Tools

Tooling is as appropriate for the member products and is not otherwise constrained by this document.

8.8 Configuration management

8.8.1 Purpose

The purpose of configuration management is to ensure an orderly, managed evolution of the entire production line – shared asset supersets, feature catalogue, and bills-of-features – over time in a way so that any version of any product can be reliably and efficiently recreated at any point in time.

The temporal management approach for feature-based PLE is built on two key principles that are independent of version management tool, repository, or shared asset type under version management control.

— All development and maintenance modifications are performed on shared asset supersets, the feature catalogue, and/or the bills-of-features.

— Product asset instances are always read-only and never modified for development and maintenance.

For temporal management in the more general ISO/IEC 26550 reference model for product line engineering and management, the application asset instances for each member product are maintained and managed separately. The temporal management strategy for feature-based PLE is focused on shared asset supersets, not the product asset instances. While modification and extension of the generated product asset instances and the optional merging of those changes back into the shared asset supersets is not included as a practice in this document, if business processes dictate, it may occur and should follow the application engineering practices defined in the more general ISO/IEC 26550 reference model for product line engineering and management.

8.8.2 Role

— PLE factory temporal management engineer.

8.8.3 Outcomes

— One or more temporal baselines for the product line. A temporal baseline is essentially a product-line-level baseline that comprises the set of baselines of each of the shared asset supersets, the feature catalogue and the bill-of-features portfolio.

8.8.4 Inputs

— Feature catalogue, stored and maintained in such a way that the current or any previous version can be recreated at any time.

— Bills-of-features, stored and maintained in such a way that the current or any previous version can be recreated at any time.

— Shared asset supersets, stored and maintained in such a way that the current or any previous version of each can be recreated at any time.

8.8.5 Tasks

8.8.5.1 Create a temporal baseline

The goal of this task is to produce a temporal baseline that corresponds to a set of member product versions. A temporal baseline includes:

— a set of individual shared asset superset baselines;

— baselines for the feature catalogue and bills-of-features that correspond in time to the set of shared asset superset baselines.

Methods should support:

— identifying the set of member products and the version of each to which this temporal baseline applies;

— identifying the version of each applicable shared asset superset that should be part of this temporal baseline;

— identifying the versions of the feature catalogue and bill-of-features portfolio, respectively, that should be part of this temporal baseline;

— capturing and representing (for later recall) this information;

— giving the temporal baseline a unique name;

— use of tooling as appropriate for the temporal baseline representation, which is not otherwise constrained by the PLE factory tools described in this document.

8.8.5.2 Use a temporal baseline to define and (re-)create a version of a member product at any time

Methods should support:

— identifying the member product and the version that should be recreated;

— identifying the temporal baseline that applies to that version of that member product;

— creating or recreating the versions of the feature catalogue, bill-of-features portfolio, and shared asset supersets that are indicated in that temporal baseline;

— using the PLE factory configurator to instantiate those shared asset supersets according to the bill-of-features, as described in 8.6.

8.8.6 Tools

Tooling for these tasks is not constrained by this document. A scheme as simple as a table or spreadsheet can suffice to store the temporal baselines (e.g. one per column) and the version number of each of the constituent parts (e.g. one per row).

8.9 Traceability management

8.9.1 Purpose

The purpose of traceability management is to ensure that product asset instances emerge from the PLE factory configurator with correct and consistent traceability links.

8.9.2 Role

— Shared asset superset engineer.

8.9.3 Outcomes

— Shared asset supersets and feature catalogue, with appropriate traceability links established in compliance with the traceability approach defined in the 8.9.4.

— Reports on traceability links in the product asset instances. For example, if the source or destination of a traceability link between product asset instances does not exist, and if this condition is considered an error, then the report shall identify these as “broken” links.

8.9.4 Inputs

— Shared asset supersets.

— Feature catalogue.

— A traceability approach specifying how shared asset supersets are linked together; for example, code to requirements, test cases to requirements, code to test cases, etc. This document has no bias with respect to this approach.

8.9.5 Tasks

8.9.5.1 Insert trace links

The goal of this task is to establish trace links, in accordance with the traceability approach, into the shared asset supersets. This is a key property of feature-based PLE: trace links are built into the shared asset supersets and carried over through configuration into the product asset instances.

Methods should support

— specifying the source and destination of a trace link in terms of shared asset superset content, such as a requirement, a unit of code, a test case, and so forth;

— reviewing the trace links for correctness and coverage;

— revising (changing) a trace link;

— use of tooling as appropriate for the trace link representation, which is not otherwise constrained by the PLE factory tools described in this document.

8.9.5.2 Check product asset instances for correctness of trace links

The goal of this task is to ensure consistent and correct trace links in the product asset instances. If, for example, a mutually inclusive requirement and test case are linked together, they are linked in their respective shared asset supersets. If one is chosen for inclusion in its product asset instance, then the other must be as well, thus ensuring that the requirement and test will be linked after configuration. If one is not chosen due to a variation point, then the other should not be chosen either. If one is chosen but the other is not, this will result in a broken link in the product asset instances, which indicates an error in the variation point specification of one or the other.

Methods should support:

— analysis for correctness of trace links;

— understanding the source of any errors found;

— reporting of any errors and their source to the factory change management authority;

— use of tooling as appropriate for the trace link representation, which is not otherwise constrained by the PLE factory tools described in this document.

8.9.6 Tools

Tooling for these tasks is not constrained by this document.

8.10 Change management

8.10.1 Purpose

This process manages changes to the feature catalogue, bills-of-features, and shared asset supersets. A change request is approved (and prioritized) or denied by a recognized change management authority. The authority can be a chartered board, or (e.g. in the early days of establishing a PLE factory) a manager whose purview includes directing changes to be made. The authority reviews a change request to assess the required effort to implement, determine whether to implement, and assign priority to the implementation. The authority plans and schedules the work, coordinating with PLE factory lead engineers and PLE factory management team (see 9.2 proviso a) 1) and d) 2)). The change request is closed after work is completed and verified by the appropriate engineering team.

8.10.2 Role

— PLE factory change control authority.

8.10.3 Outcomes

— Adjudicated (approved/prioritized, denied) change requests.

— Feature catalogue, bills-of-features, and shared asset supersets are updated as appropriate.

— Work to implement an approved change request is planned and scheduled.

— Closed change request, which is closed after rejection, or (in case of approval) after work to implement it is completed and validated.

8.10.4 Inputs

— Change requests.

— Feature catalogue, bills-of-features, and shared asset supersets.

8.10.5 Tasks

8.10.5.1 Initiate a change request

The purpose of this task is to create a change request to the feature catalogue, bills-of-features, and/or shared asset supersets.

Methods should support:

— creating a change request using a standardized format;

— storing the change request for later recall or revision;

— inserting the change request into a workflow process that leads to its adjudication;

— use of tooling as appropriate for change request management, which is not otherwise constrained by the PLE factory tools described in this document.

8.10.5.2 Adjudicate a change request and capture rationale for the adjudication

The purpose of this task is to assign an initial state to a change request, such as “rejected” or “accepted”.

Methods should support: