Building Software Process Line Architectures from Bottom Up

Hironori Washizaki

National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
washizaki@nii.ac.jp

Abstract. In this paper, we propose a technique for establishing process lines, which are sets of common processes in particular problem domains, and process line architectures that incorporate commonality and variability. Process line architectures are used as a basis for deriving process lines from the perspective of overall optimization. The proposed technique includes some extensions to the Software Process Engineering Metamodel for clearly expressing the commonality and variability in the process workflows described as UML activity diagrams. As a result of applying the proposed technique to hardware/software co-design processes in an embedded system domain, it is found that the proposed technique is useful for defining consistent and project-specific processes efficiently.

1 Introduction

Process tailoring is an approach for defining project-specific processes by adding, removing or modifying the activities and the required inputs/outputs of a base process model to develop high-quality system/software efficiently. Project-specific processes are a collection of interrelated, concrete activities along the time line of the project, which take into consideration the characteristics of the specific project. Conventional tailoring approaches can be divided into two major types[1]: component-based approaches and generator approaches. The former tries to build a project-specific process based on existing process parts, but it lacks a way to address the overall compatibility and consistency of the derived processes. The latter tries to build a project-specific process by instantiating a typical process architecture, but it lacks a way to reuse process fragments.

In this paper, we propose a new process-tailoring technique which solves the problems with component-based and generator approaches by building a *Process-Line Architecture* (hereafter *PLA*) and deriving project-specific processes from the PLA. A process line is a set of similar processes within a particular domain, and is an application of the idea of product lines to processes. Process lines were proposed by Romback[2] and Jaufman[1], but parts of the definition and technical system are still not well-defined, and not sufficient for creating a concrete framework. Other similar ideas have also been proposed, including process libraries[3] and families[4]; however, these are not always oriented toward overall optimization, and do not lead to generally-applicable process-model structures.

2 Process Line Architecture

We define a *Process Line* as "a set of processes in a particular domain or for a particular purpose, having common characteristics and built based upon common, reusable process assets (such as PLAs, requirements)". The relationship between process lines and PLA is shown in Figure 1.

A PLA is "a process structure which reflects the commonality and variability in a collection of processes that make up a process line from the perspective of overall optimization". We mean "overall optimization" as preparing a PLA with general utility rather than defining separate but similar optimized processes. By deriving individual process from the PLA, the fixed amount of additional effort required in the future can be reduced, and timeliness of completion can be improved. Commonality in a PLA is represented by the core process, which is made up of the common parts of the set of processes. Variability is represented by the variation points and process variants. Variation points are activities (or the inputs/outputs or roles that effect activities) which can be changed according to the characteristics of a specific project. Process variants are the concrete candidate activities (or inputs/outputs, etc.) that are applied to the variation points. Processes that are specialized for a particular but similar project can be defined and applied effectively by combining, extending and reusing the core process and variants in a particular problem domain.

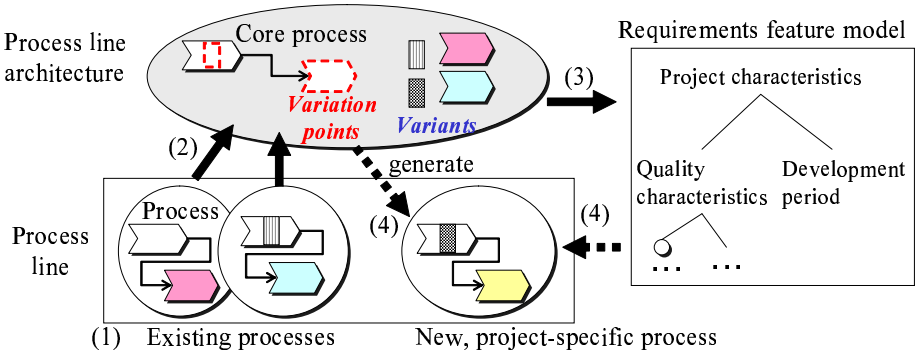

Fig. 1. Process line framework and bottom-up building activities

It is difficult to adequately analyze commonality and variability in a domain from scratch without missing anything; this is to say a "top-down" approach. So we propose the following "bottom-up" technique (shown in Figure 1) for building a PLA using existing knowledge on process definitions and applications in the well-known problem domain. We define *Process Line Engineering* as "a system of interrelated strategic and systematic approaches for building, applying and managing process lines". Based on this concept, the following activities (1)–(3) are in the domain engineering, and (4) is in the application engineering.

(1) Several existing processes in the selected problem domain are gathered together. These processes, sharing common parts, can be combined to form the process line P.

(2) Commonality of P (the gathered collection of process) is defined as the core process including variation points. Variability is defined as a set of variants defined for each variation point in the core process. In describing below how the PLA is built by our technique, we use some new, original extensions to the Software Process Engineering Metamodel (SPEM[5]) to clearly express the commonality and variability in the process workflows. These cannot be expressed with traditional SPEM. The procedure is described as follows:

(a) Make the smallest process in P a core process, p_c. Then apply (b) to all of the remaining processes in P ($p_i \in P - \{p_c\}$). We assume that p_c's workflow is composed of a set of interrelated activities and conditional branches along the time line, denoted as $p_c = e_{c1} \rightarrow ... \rightarrow e_{ck} \rightarrow ... \rightarrow e_{cm}$. Similarly, we denote the p_i's workflow as $p_i = e_{i1} \rightarrow ... \rightarrow e_{ij} \rightarrow ... \rightarrow e_{in}$.

(b) All activities and conditional branches e_{ij} in the p_i's workflow are compared with all elements in the p_c workflow, e_{ck}. If these elements are not the same, the following (c)–(g) are performed. The sameness, specialization and generalization relation between two process elements can be identified by comparing the activity details, pre/post-conditions, inputs/outputs, roles, and environments including tools. In our technique, the above-mentioned comparison is conducted manually. As our future work, we will try to use tool-supported techniques such as a technique proposed by Ocampo [6].

(c) If e_{ij} is a specialized element of e_{ck}, we create the generalization relationship denoted as $e_{ck} \triangleleft\!\!-\!\!e_{ij}$, label e_{ck} with a $\ll$`variationPoint`$\gg$ stereotype, label e_{ij} with $\ll$`variant`$\gg$, and add e_{ij} to p_c. Conversely, if e_{ck} is a specialized element of e_{ij}, exchange e_{ij} for e_{ck} in p_c and perform the same way.

(d) If there is no element which specializes or generalizes e_{ij}, and the element preceding e_{ij} (i.e. e_{ij-1}) on the p_i's workflow is equal to e_{cl} in p_c, set a transition from e_{cl} to e_{ij}. Label e_{ij} with an $\ll$`optional`$\gg$ stereotype, and add e_{ij} to p_c. When actually defining a concrete process with a selection of the above-mentioned optional element, we will proceed the element e_{cl+1} after proceeding from e_{cl} to e_{ij}. In other words, the obtained process architecture with the optional element provides two different workflow definitions: $e_{cl} \rightarrow e_{cl+1}$ or $e_{cl} \rightarrow e_{ij} \rightarrow e_{cl+1}$.

(e) If there is no element which specializes or generalizes e_{ij}, and the element preceding e_{ij} (i.e. e_{ij-1}) already have the $\ll$`variant`$\gg$ or $\ll$`optional`$\gg$ stereotype, set a transition from e_{ij-1} to e_{ij} and add e_{ij} to p_c.

(f) If a $\ll$`variant`$\gg$ or $\ll$`optional`$\gg$ element was added in (c) or (d), add transitions to appropriate elements within p_c for each of that element's original transitions. When doing this, if there are two or more outgoing transitions for one element, not including conditional branch elements, draw a dashed line over these transitions and annotate the line with a constraint {`xor`} to show clearly that one transition must be selected when the concrete process is defined.

(g) If a $\ll$`variant`$\gg$ or $\ll$`optional`$\gg$ element e_x was added in (c) or (d), and e_x requires that other $\ll$`variant`$\gg$/ $\ll$`optional`$\gg$ elements (e_y) must be preceded, add a dependency relationship from e_x to e_y, denoted as $e_x \overset{\ll requires \gg}{\cdots >}$

e_y. These dependency relationships and the exclusive selection relationships described above are important for maintaining process consistency.

(3) The project characteristics as predictable requirements for a process line are defined corresponding to the commonality and variability built into the PLA. Feature Diagram[7] can be used to define the requirements that accompany the commonality and variability. Feature diagrams are a way of expressing requirements having both variability/commonality and consistency, by allowing substitution and selection of logical units called features, which are functional or qualitative requirements.
(4) By reusing the PLA derived through the above procedure and the requirements including commonality/variability for the process line, project-specific processes that maintain consistency can be defined efficiently. For example, if the requirements on the process line are expressed by a feature diagram, and the part of the PLA which handles each feature is recorded (i.e. there is traceability between PLA and feature diagram), a customized consistent process can be derived easily by selecting features. Moreover, the PLA can be used as a basis for comparing similar processes.

3 Application to Hardware/Software Co-design

As an example, we consider building a process line for hardware/software co-design process in embedded system development. When defining this process, it is necessary to decide, on a per project basis, variations like when the hardware architecture specification will be decided, and whether the division and mapping of specifications will be iterated. As such, we tried building a PLA from the bottom up as described in the previous section.
(1) As representative but partially different processes for hardware/software co-design, we collected the Wolf process (denoted as p_W[8]), the Axlesson process (p_A[9]), and the process from the Kassners (p_K[10]). The workflow of each process is shown with an activity diagram based on SPEM in Figure 2. Due to space limitation, roles and inputs/outputs have been omitted from each diagram.
(2) The PLA workflow derived from the analysis for variability and commonality in these three processes is shown on the left side of Figure 3. Figure 3 clearly shows the core process with variation points, variants, optional elements, and exclusive transitions. For example, an activity `‘‘Specification definition’’` in the core process is labeled as a variation point, and can be substituted with a variant `‘‘Executable behavior specification definition’’` that expresses more detail. In addition, there are several conditional branches that implement iteration cycles in the process; these are optional elements.
(3) The project characteristics were analyzed as requirements corresponding to the variability/commonality in the PLA. A feature diagram on the right side in Figure 3 shows the result. We have related the optional and substitute features to the optional elements and variants in the PLA.
(4) Using the resulting PLA and feature diagram, various processes including the original three processes can be derived in a consistent and efficient way

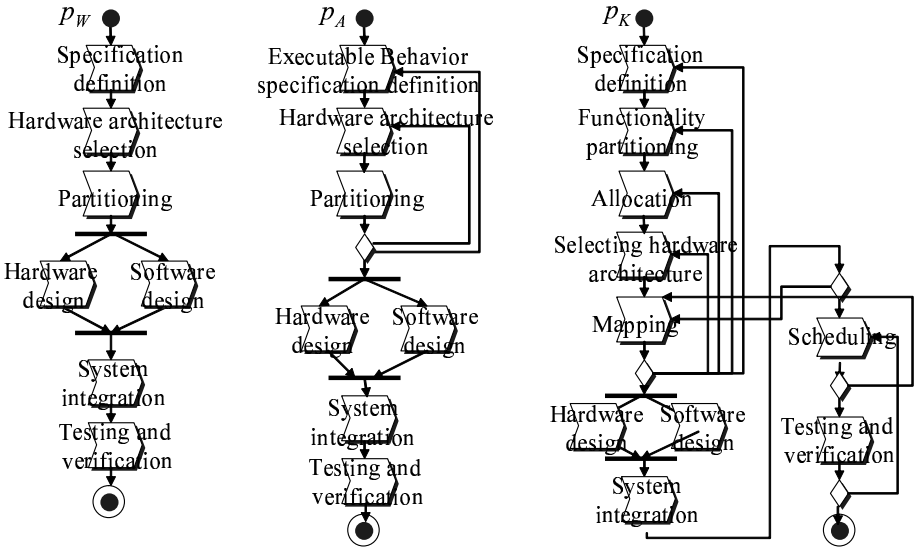

Fig. 2. Collected process workflows

based on the requirements. For example, for a short-term project where the decision on hardware specifications is late, and performance and reliability might be sacrificed due to an extremely short development period, we will simply select the ‘‘`Late`’’ feature on the feature diagram. This defines a process where iteration cycles are excluded, and the ‘‘`Hardware architecture selection`’’ activity is done after ‘‘`Allocation`’’. This newly defined process is consistent; for example, ‘‘`Functional partitioningn`’’ will be included and located before ‘‘`Allocation`’’ according to the dependence relationship. Without using a PLA and feature diagram, it would not be easy to quickly define a similar, new and consistent process based on various project characteristics.

4 Conclusion and Future Work

In this paper, we have defined terminology and a framework for the development of process lines, and shown a technique for building practical process-line architectures that allows consistent, project-specific processes in a given problem domain to be defined efficiently. Also, in order to express the commonality/variability in PLA workflows, we have proposed a notation which is an extension to SPEM. Finally, by building a PLA and feature diagram for hardware/software co-design process, we showed that consistent project-specific processes can be derived easily based on the proposed technique. In the future, we plan to explicitly handle factors such as resource limitations, inputs and outputs, and pre- and post-conditions in the proposed technique.

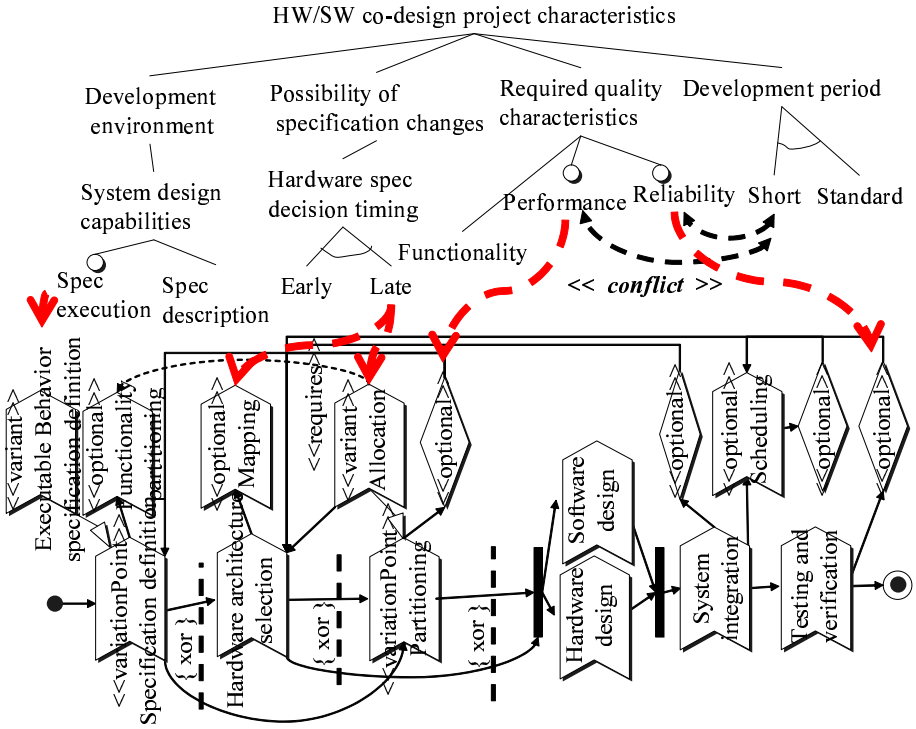

Fig. 3. Obtained co-design process line architecture and its feature diagram

References

1. O. Jaufman and J. Munch: Acquisition of a Project-Specific Process, Proc. 6th International Conference on Product Focused Software Process Improvement, 2005.
2. D. Rombach: Integrated Software Process and Product Lines, Post-Proceedings of the Software Process Workshop 2005, LNCS Vol.3840, 2005.
3. P. Mi et al.: A Knowledge-based Software Process Library for Process-Driven Software Development, 7th Knowledge-Based Software Engineering Conference, 1992.
4. Y. Matsumoto: Japanese Software Factory, in Encyclopedia of Software Engineering, (ed.) J.J. Marciniak, John Wiley & Sons, 1994.
5. OMG: Software Process Engineering Metamodel Specification, Version 1.1, 2005.
6. A. Ocampo, R. Bella and J. Munch: Software Process Commonality Analysis, Software Process Improvement and Practice, Vol.10, No.3, 2005.
7. J.C. Trigaux and P. Heymans: Modelling variability requirements in Software Product Lines, Technical report PLENTY project, 2003.
8. W.H. Wolf: Computer as Components: Principles of Embedded Computing System Design, Morgan Kaufmann, 2001.
9. J. Axelsson: Hardware/Software Partitioning of Real-Time Systems, IEE Colloquium on Partitioning in Hardware-Software Codesigns, 1995.
10. K.C. Kassner and K.G. Ricks: Hardware/Software Co-Design of Embedded Real-Time Systems from an Undergraduate Perspective, Workshop on Computer Architecture Education, 2005.