Integrating Process, Product, and People Models to Improve Software Engineering Capability

J. Scott Hawker
University of Alabama
hawker@cs.ua.edu

Abstract

We present a software engineering model that integrates three important elements: software product components, software process components, and people. We focus on ways these three elements should be integrated to provide a foundation for process execution and product engineering tools that improve software engineering capability.

1. Introduction

We seek to provide tools that enable people performing software engineering processes to produce software products that are of value to their customers. A key enabler of these tools is an underlying model that integrates people, processes, and products. The tools instantiate, navigate, and manipulate this integrated model. Tools based on such a model will significantly enhance the engineering capability of organizations developing large-scale, long life-cycle products from reusable and evolving software components and product-line architectures.

1.1. A scenario

Consider a new project at a software development organization called BiggerSoft. A small software engineering team receives a task to develop a module in a larger system. BiggerSoft uses a process execution environment, PXE, which guides the developers through the software engineering process and captures the results. PXE gives the developers access to artifacts that scope their task, including the overall system vision document, system requirements, system architecture, and the initial draft of the requirements for their module. PXE suggests that they begin a pair of activities, one to further develop the requirements and one to define a feasible software architecture for their module. PXE provides guidelines, document templates, examples of other requirements and architectures, relevant architecture styles and design patterns, and additional guidance to aid the developers in carrying out their tasks.

PXE suggests tasks and guidance that are consistent with BiggerSoft's adopted software engineering processes, and it allows process tailoring and configuration to meet the specific needs of the project at hand. The developers tailor and accept these activities and the corresponding types of product artifacts and engineering techniques they will use. PXE then instantiates and initializes objects that capture these new process activities and provides facilities for capturing activity progress, risks, and issues.

PXE is integrated with PDE, the integrated product development environment for BiggerSoft's developers. Indeed, PXE and PDE are one system of tools for software engineering. So, in addition to instantiating process artifacts, PXE also instantiates product artifacts in PDE to capture the evolving product models that will result from executing the development process activities, and it registers these product objects with the PDE configuration and change management tools.

As development proceeds, PXE allows the development team to evolve their product artifacts, capture process progress, enact new process activities (such as document review) and sub-activities (such as detailing a use-case description to capture detailed requirements or developing a prototype to investigate the use of a new technology), and make all this information available to the development team and project stakeholders.

Over time, PXE/PDE captures a rich, integrated history of process activities and product models for the newly developed software module. Even after the module is tested, integrated with the rest of the system, and transitioned to customer operation, PXE manages the activities and artifacts of product support, maintenance, enhancement, and retirement.

Because BiggerSoft's software engineering processes include asset harvesting and reuse, PXE provides tools and guidance to support these activities. The integrated history of process activities and product artifacts is a powerful enabler of these tools. When a developer is

looking for existing solutions to use in a new project, they may discover something relevant by searching based on product characteristics such as similar requirements statements, design patterns, or implementation technologies, or based on process characteristics such as similar development techniques, customer profiles, development team members, or any other criteria.

Given a potential reuse match, the developer can navigate to related information, for example, to identify re-engineering tasks, to estimate development cost based on the cost of previous work, to view defect reports and fixes, to identify the engineer who designed the prior product so they can be consulted on design modifications, and to see what other projects have also used or considered using the artifacts. The PXE can provide guidance on activities and techniques for reusing the artifacts and suggest investing additional effort to re-engineer and package the artifacts for subsequent in-house reuse or external sale.

BiggerSoft recognizes that their software product artifacts are valuable assets. They have invested in PXE/PDE to provide the processes, tools, and techniques for capturing and reusing software components and product-line architectures. BiggerSoft also recognizes that the development process, itself, is a valuable asset, and they have developed processes, tools, and techniques for capturing the software engineering process activities as reusable process components that can be tailored and assembled into specific development processes that meet the needs of their development projects. So, PXE also provides tools and guidance to support and manage process-engineering projects that develop, evolve, and reuse process components and complete processes. BiggerSoft views their process components and processes as first-class assets, receiving all the same attention and management control as their product components and delivered products.

Figure 1 illustrates a portion of this scenario and previews some of the elements of our integrated process, product, and people model. The next sections discuss the current state of technology in modeling, and then present our model and the ways it integrates the three key areas.

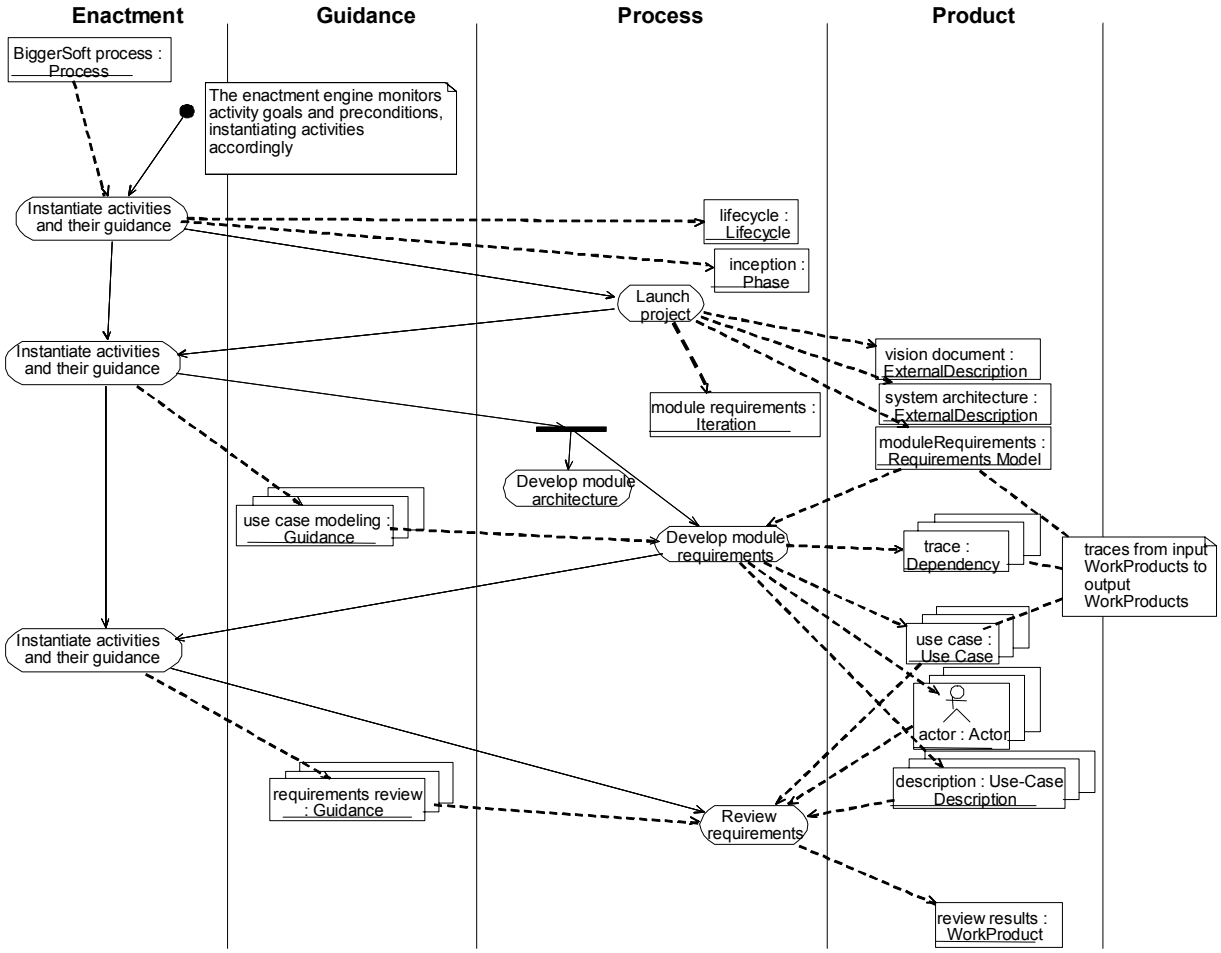

Figure 1. A portion of a development scenario supported by an integrated process and product model

2. Available modeling technology

Software engineering researchers have made significant progress in three important areas: component-based software products, component-based processes for software engineering, and technologies and business practices for software component reuse. Research and practice in some of these areas have reached a point of maturity such that international standards are available which stabilize and unify the field, allowing the integration and interoperability of standard-conforming products. Our research seeks to integrate the results in these three areas into a synergistic, integrated model supporting the needs of organizations developing large-scale, long-life software.

Product Components. There are mature software technologies, architectures, and patterns for distributed, component-based software products [1, 2]. There is an active marketplace of products based on these technologies, such as, Sun's Java execution environments [3], Microsoft's COM+ and .NET environments [4], and the Object Management Group's CORBA family of standards [5]. There are also powerful component definition, design, and implementation environments for building and managing component-based software throughout its lifecycle. Widely-used commercial products are available from Rational (www.rational.com), TogetherSoft (www.TogetherSoft.com), and others.

The Object Management Group (OMG) has led the standardization effort by providing CORBA, UML, CORBAcomponents, and related standards in their Model-Driven Architecture (MDA) initiatives [6]. Complementing these formal, component-based product standards are the *de facto* standards represented by the widespread adoption and use of the Microsoft COM+, and .NET specifications [7, 4] and the Sun Java2 specifications [3]. In addition, software development organizations are benefiting from the standardization and adoption of domain-specific specifications of component-based frameworks [8].

Process Components. Software engineering processes can be characterized as assemblies of engineering activities. The activities are presented as reusable process components, defining and encapsulating the process steps a software engineer performs to develop software. Software process engineers form a specific software engineering process by selecting, tailoring, and assembling process components into engineering workflows, matching the needs of a software development project to the practices and maturity of the software development organization.

The recently-adopted OMG Software Process Engineering Metamodel (SPEM) [9] provides an international standard for process engineering models. The standard is complemented by Rational's Unified Process [10] and the OPEN consortium's (Object-oriented Process, Environment and Notation) OPEN Process Framework (OPF) [11]. Our models closely align with SPEM, but further work is necessary to align them with the most recent SPEM release.

Software Component Reuse. There is a growing development and practice of software asset management and business practices that focus on asset harvesting and reuse, organized around evolving product-line architectures [12, 13]. The results are strong and promising [14, 15]. In addition, the shift in business practices toward process-centered organizations [16], agile and virtual enterprises [17], and collaborating and learning organizations [18] provide us with new ways to model and understand the "people side" of software engineering.

2.1. Our research

Our research seeks to integrate the results in the three areas, above, into a synergistic, integrated model supporting the needs of organizations developing large-scale software. We strive to identify the concepts and architectures that enable software tools and methods that these organizations can use to manage their assets: their software intellectual property and products, their software developers with development tools and techniques, and the software engineering processes that guide the organization's use of these assets to continuously produce and deliver customer value.

In this paper, we introduce an integrated product and process component model based on the OMG UML and SPEM models. We focus on the many ways that product and process elements are related, and we define views of the integrated model that enable and support some of the capabilities described in the scenario of Figure 1.

We are developing an experimental process execution environment to validate our concepts and to exercise the new standards. We are eager to help enable the integration of product development environments, process engineering environments, and process execution environments into a powerful suite of tools that improve software engineering capabilities.

3. Conceptual model

A software development project involves people carrying out engineering process activities to produce software products. Figure 2 illustrates the relationship between the "four Ps" of software engineering activities [10].

Project: People + Process → Product

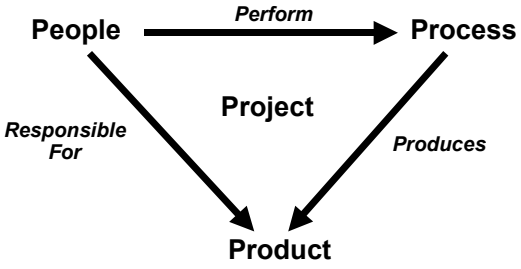

Figure 2. The Four Ps of software engineering

Figure 3 illustrates that a common, integrated model will support extracting discipline-specific views of software development projects that span time and space (geographical, organizational, and functional space). The integration of product, process, and people models allows the process and people models to parallel and complement the rich structure of the product models. For example, engineering process activities and techniques align with product models that feature component-based architectures, product-line architectures, component reuse and evolution, and the use of architecture and design patterns and frameworks. The composition of the process activities reflects the composition of the product models. Further, the integration of product, process, and people models facilitates reuse management by allowing product developers and project managers to navigate through the history of product artifacts, tracing to requirements, design and test models, version history, other reuse contexts, responsible individuals and organizations, and various cost and quality metrics captured as part of prior development activities. By formally representing all these product, process and people model elements and embedding them in a configuration and change management system, the software developer can readily reuse and manage the growing wealth of product and process component artifacts available to an organization. With the right information, they can better make the right decisions and deliver cost-effective, high-quality products.

Given the huge scope of the model and the diversity of stakeholders using the model, we identify numerous viewpoints of the model that focus on the roles and understanding of specific stakeholders. The viewpoints include guidance on how to interactively extract and browse specific model views and how to navigate among the many relationships in the highly-integrated model.

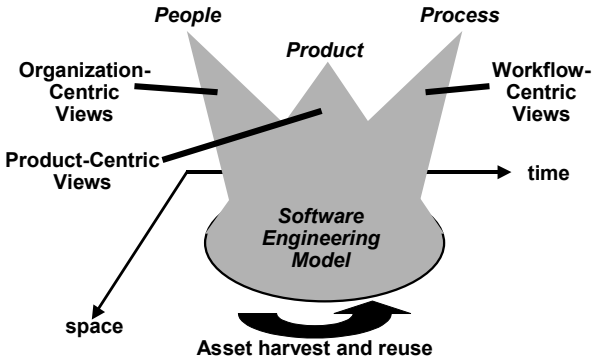

Figure 3. Views of an integrated software engineering model

Figure 4 uses a UML class diagram to illustrate the essential structure of the integrated model, aligned with the Four Ps of Figure 2. Figure 4 shows that people have roles in the software engineering process. In those roles, they are responsible for work products. They perform process activities to produce work products, using other work products as input to the process activity. Note: we use the People-Process-Product triangle layout in our other diagrams, helping to quickly identify the alignment of classes and Four P model aspects.

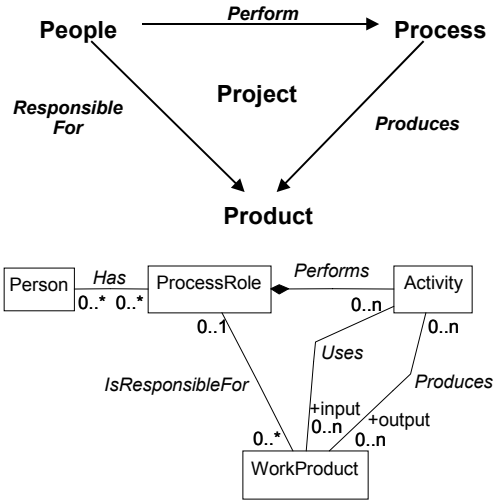

Figure 4. UML class diagram capturing the essential structure of the integrated model, aligned with the Four Ps model

Figure 5 shows the organization of the integrated model as packages. The three key packages capture the process, product, and people models.

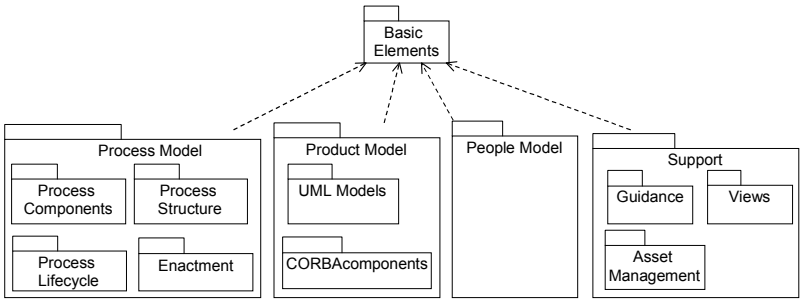

Figure 5. Package structure of the integrated model

4. The integrated model

The following sections describe the models in the packages of Figure 5, focusing on the integration of the process, product, and people models.

4.1. Basic Elements package

The Basic Elements package provides the basic conceptual classes for the overall model. Figure 6 shows the package contents. The "root" of all model elements is the abstract ModelElement class. Each ModelElement has an external description that provides a human-readable description of the element. Each model element can also have associated Guidance, which provides further information to those people or tools using the model element. The OMG SPEM defines types of Guidance for process elements, including Techniques, Guidelines, Estimates, Templates, etc. We define additional guidance, including guidance for product and people elements, as Figure 7 suggests.

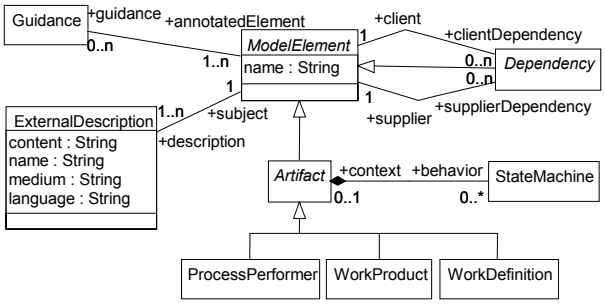

Figure 6. Basic Elements package

The Dependency specialization of ModelElement defines binary relationships between ModelElements. For example, the SPEM defines types of dependencies between process elements, including Precedes and Trace. We also define other types of Dependency for and between process, product, and person role elements; Dependency is our general-purpose relationship modeling tool.

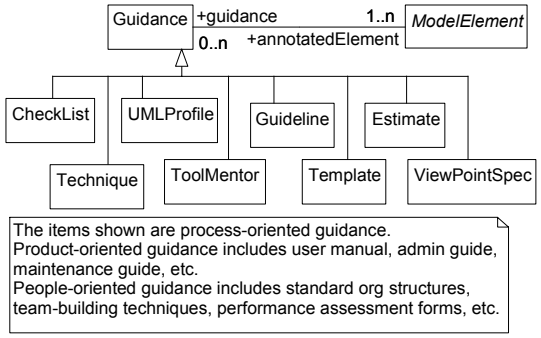

Figure 7. Types of Guidance

Although we usually think of work products as the artifacts of interest in software engineering models, we also identify process elements (WorkDefinition) and person role elements (ProcessPerformer) as artifacts. This emphasizes that all three of the element types are first-class elements of the integrated model, and it allows the elements to be under the control of an asset management system to track configuration and changes. Artifacts have an associated StateMachine. This allows capturing work products in various states of completion, or processes in various states of execution, or people with differing states of assignment and skills. Here, StateMachine refers to the UML state machine and action model, bringing the functionality of state-dependent behavior, activity graphs, action semantics, workflows, process enactment, and other functionality to support executing software engineering processes.

4.2. Product package

The Product package provides for the representation of product models and dependencies between them. The primary sub-package within the Product package is the UML Models package, portions of which are illustrated in Figure 8. Here, we adopt by reference the models in the UML standard specification [19]. The UML specification defines numerous ways that large, complex

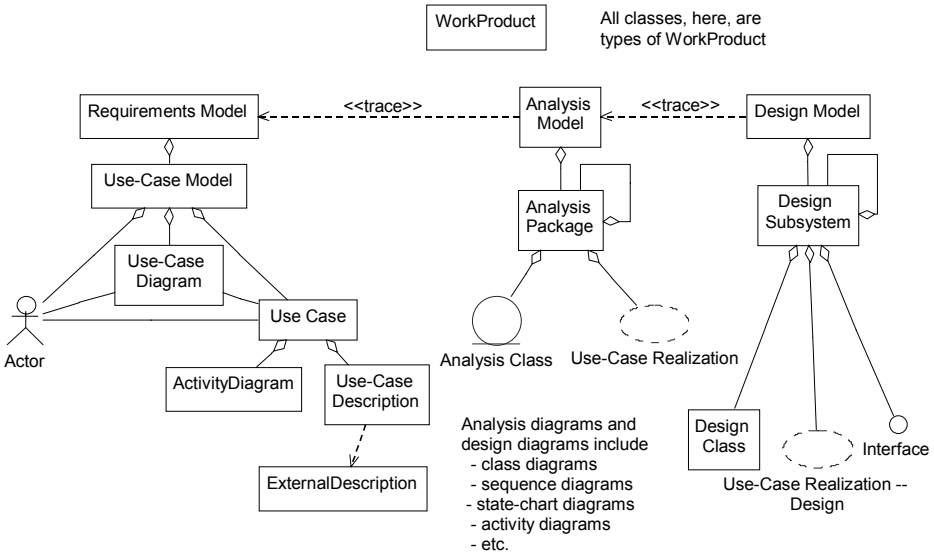

Figure 8. Portions of the UML Models package

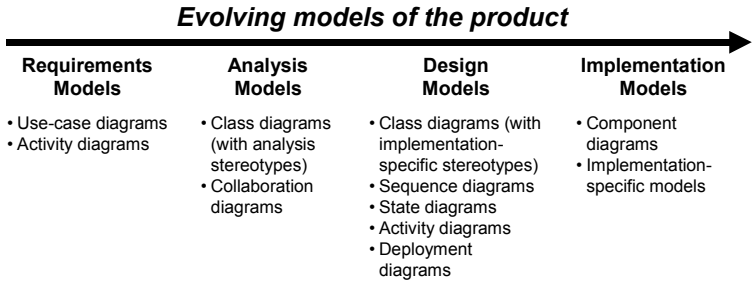

Figure 9. Product models as an evolution of UML models

software products can be represented as UML models, including class diagrams, interaction diagrams, component diagrams, state diagrams, etc. There are myriad books and guides on using UML to model software products in all phases of development.

To help organize the UML product models and align them with process models, we adopt the Unified Process [10], which characterizes the product models as an evolution of UML diagrams representing Requirements Models, Analysis Models, Design Models, and Implementation Models, as Figure 9 shows. These models, together with supporting documents, capture an evolving syntactic and semantic description of the software product during development.

Because we emphasize component-based software reuse in our model, the product model also includes the OMG CORBAcomponents model [5]. CORBAcomponents represent reusable product components defined in terms of the interfaces the component provides and uses and the events the component publishes and consumes. We also assume the corresponding CORBAservices component infrastructure of component containers and their use of standard communication (object request broker), notification, persistence, transaction, and security services [20].

As a development organization adopts component-oriented architectures, architectural styles, domain-specific product-line architectures, design patterns, and other repeating product structure, the family of product models for the organization will develop a very rich structure. As we align process and people models to the product models, the structure of the process and people models will reflect the structures of the product models. For example, certain groups in an organization will be responsible for developing specific architectural components, or there will be design process guidelines for

specializing each component of a product-line architecture into components of a product-specific architecture.

4.3. Process Structure package

Figure 10 illustrates the Process Structure subpackage of the Process package. This is the main place for integrating the process model (Activity and WorkDefinition) with the product model (WorkProduct) via people (ProcessRole and ProcessPerformer). A ProcessRole is responsible for a set of WorkProducts and a ProcessRole performs Activities. A ProcessPerformer is the performer of a group of WorkDefinitions that cannot be associated with specific ProcessRoles. WorkDefinitions use and produce a partially-ordered collection of WorkProducts. The WorkProducts are inputs to and/or outputs from the corresponding WorkDefinition. The attribute 'hasWorkPerArtifact' on the ActivityParameter indicates that multiple instances of WorkDefinition are needed, with one WorkDefinition per instance of the corresponding WorkProduct.

In the integrated model, the work definition and work product aggregation structures often parallel each other: there are subwork definitions for activities to produce corresponding subwork products, and this subwork is assigned to a specific subgroup in the development team. Also, the Process Lifecycle package (defined in a later section) defines Phase, Iteration, and Lifecycle as subtypes of WorkDefinition. These process lifecycle elements often correspond directly with work products identified as deliverables of the corresponding phase, iteration, and lifecycle. Using the Dependency relations of ModelElements in Figure 6, numerous other parallel structures can be modeled. For example, a design process activity following an analysis activity can correspond to a design product model tracing to an analysis model, where the analysis model is an input work product to the design activity and the design model is an output work product from the design activity. Model viewpoints will exploit these and other Dependency relations between WorkProduct, WorkDefinition, and ProcessRole.

4.4. Process Component package

Figure 11 shows the Process Component subpackage of the Process Model package. This package provides the basic structure to collect process elements into reusable process components. A ProcessComponent is an internally consistent and complete collection of WorkDefinition artifacts and associated guidance, kinds of WorkProducts, etc. A ProcessComponent can be combined with other ProcessComponents to assemble a partial or complete software engineering process.

Disciplines partition the Activities in a ProcessComponent into related activities. For example, the Unified Process identifies groupings of activities into core workflows such as Requirements, Analysis, Design, Implementation, and Test. The composition of Activities in the Discipline parallels the composition of product model artifacts in the WorkProduct output from the Activities. The Lifecycle Precondition and Goal Constraints on the Activity WorkDefinition assure that software engineering activities reflect the 'trace' and other dependencies between WorkProducts.

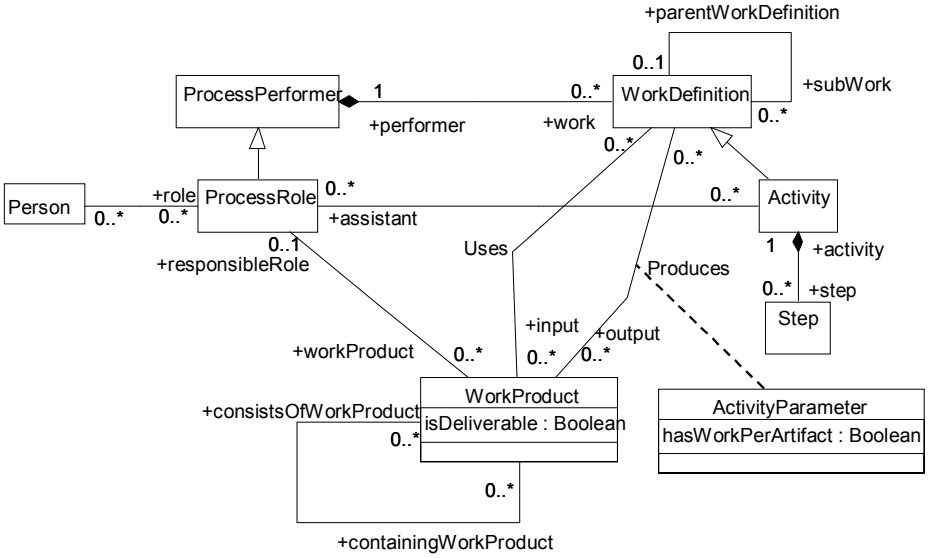

Figure 10. Process Structure package

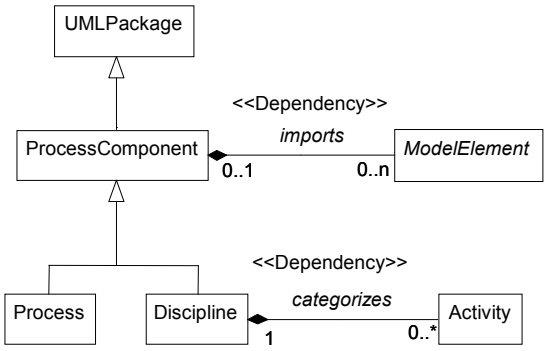

Figure 11. The Process Component package

4.5. Process Lifecycle package

The Process Lifecycle subpackage of the Process package captures the logical (often temporal) ordering of process activities. This supports the enactment of partially-ordered sequences of activities in workflows. Figure 12 illustrates the Process Lifecycle package. The Enactment subpackage of the Process package captures the workflow and enactment behavior of process activities, that is, the workflow engine creates WorkDefinition instances in response to posted Goals and enabled Preconditions. The constraint for a WorkDefinition (or Activity) is typically defined in terms of conditions on its input and output WorkProducts. This is an essential integration point between process and product models. For example, a goal for a subsystem testing activity might be that the subsystem is in a "tested" state, and a precondition of that testing activity might be that all the code WorkProducts in the subsystem be in a "unit tested" state.

A Lifecycle is a sequence of Phases, which are, in turn, a sequence of Iterations. A Lifecycle defines the complete process to be performed in a given project. The Phases are strictly ordered in time with no overlap. To further integrate the process and product models, we adopt structure from the Unified Process so that each Phase has guidelines capturing the desired quality of specific product models. The desired quality provides constraints on Phase WorkDefinitions that use and produce WorkProducts. The quality of a given WorkDefinition is interpreted by people and recorded by the enactment engine.

For example, the Unified Process Inception Phase (an instance of a Phase WorkDefinition) has goals that correspond to the existence of some percentage of requirements and analysis models (represented as UML diagrams and text) to limit project scope, and the existence of some percentage of design (again, represented as UML diagrams and text) to capture architecture feasibility. Satisfying these goals can be interpreted as satisfying preconditions for the Unified Process Elaboration Phase, which follows the Inception Phase.

Note that each enactment of a process phase results in baselining the associated product models using the asset management facilities from the Support package. The logical ordering of process phases corresponds to a 'trace' dependency ordering of product model versions, captured by the WorkProduct input/output relations to the WorkDefinitions. Thus, the structure of the process model is reflected and traceable through the 'trace' dependencies of the product model.

Note also that adopting a methodology, via a selected and tailored process model instance, enables the capture of rich relationships for subsequent reuse analysis. A process execution tool can record the activities, the work products, the guidance used, the people involved, the pre/post conditions, etc.

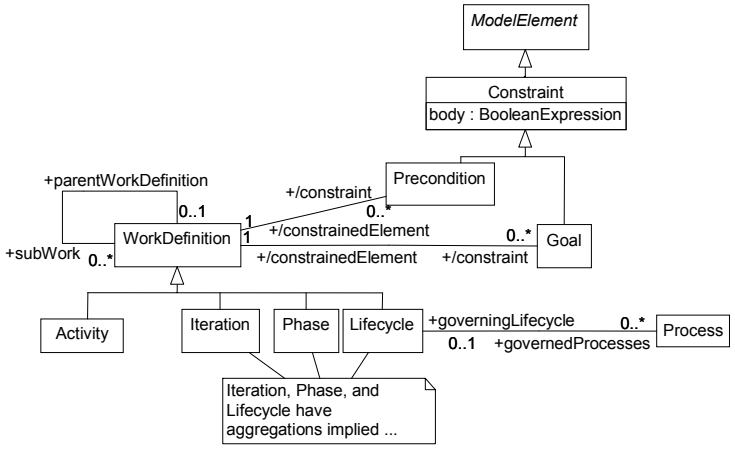

Figure 12. Process Lifecycle package

4.6. Support package

The Support package in Figure 5 includes Guidance, Asset Management, and Views. The section on the Basic Elements package and Figure 7 discussed Guidance. Asset Management captures all ModelElements as ConfigurationItems with corresponding Versions, VersionHistories, and Variants. A Configuration is a selection of a specific Version for each ModelElement in the Configuration. Because all ModelElements are ConfigurationItems, Configurations can include product, process, people, and support model elements.

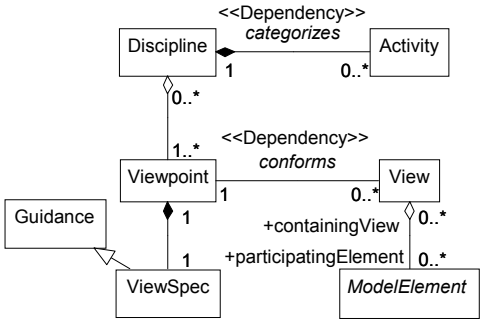

Figure 13. Views package

The Views subpackage of the Support package provides ways to specify and create views of the integrated process, product, and people model. Figure 13 Illustrates the Views package, which is motivated by the IEEE Architectural Description standard [21]. The intent is to provide discipline-specific views into instances of the integrated product, process, and people model. The views focus on the types of model elements and relationships of interest to a given software engineering discipline. Viewpoints are templates for the kind of views available for a Discipline, and ViewSpecs define how to construct a View for a given Viewpoint.

For example, views supporting a product component designer would focus on artifacts in design models, activities for designing components that interface with this component, guidance for designing product components, trace relationships to the requirements and analysis models that the design realize, and trace relationships to implementations and tests for the design. While viewing any given model element instance, the designer could navigate to any other model element via the relationships defined in the integrated model. The designer could also navigate the model definition (the metamodel) to understand the semantic meaning and intent of model instances and relationships.

5. Conclusions and next steps

We have formed a comprehensive model that integrates existing techniques and standards for modeling software products, processes, and people, and we have analyzed the model to identify the key relationships that integrate the three aspects. Our on-going effort is focused on validating our models by building and using process execution tools that instantiate, manipulate, and browse the integrated model. We are also developing product and process reuse scenarios and the discipline-specific viewpoints to improve a project manager's ability to estimate and manage reuse-based projects.

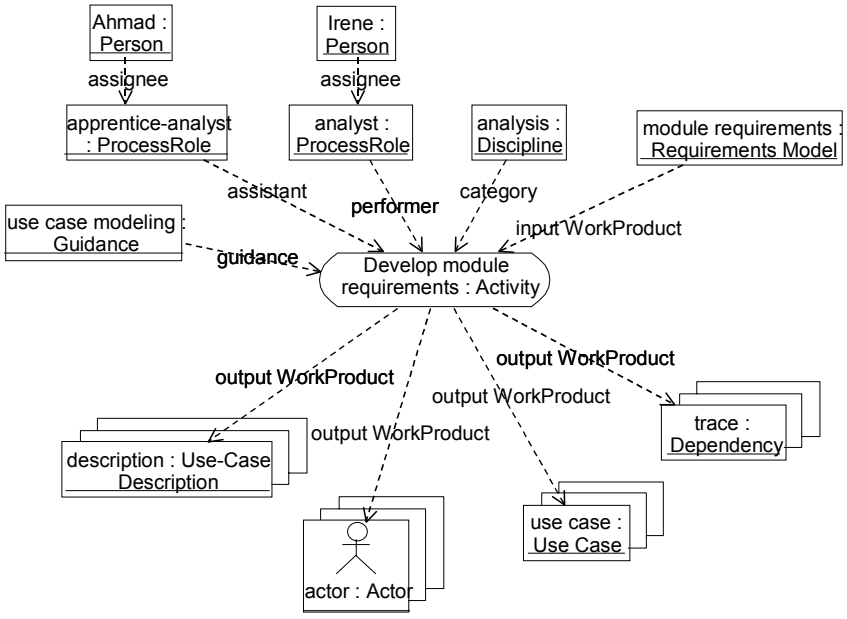

Figure 14. A view for the analysis discipline focusing on the requirements activity of Figure 1

6. References

[1] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, Harlow, England, 1998.

[2] G.T. Heineman, and W.T. Councill, *Component-Based Software Engineering: Putting the Pieces Together*, Addison-Wesley, Boston, MA, 2001.

[3] *Java 2 Enterprise Edition Specification v1.3*, Sun Microsystems, Palo Alto, CA, 2001.

[4] D.S. Platt, *Introducing Microsoft .NET*, Second Edition, Microsoft Press, Redmond, WA, 2002.

[5] *CORBA Components*, Version 3.0 (document number formal/02-06-65), Object Management Group, Framingham, MA, 2002.

[6] *Model-Driven Architecture (MDA)*, (document number ormsc/2001-07-01), Object Management Group, Framingham, MA, 2001.

[7] D.S. Platt, *Understanding COM+*, Microsoft Press, Redmond, WA, 1999.

[8] M.E. Fayad and R.E. Johnson, editors, *Domain-Specific Application Frameworks: Frameworks Experience by Industry*, John Wiley & Sons, 1999.

[9] *Software Process Engineering Metamodel Specification* (document number ptc/2002-05-04), Object Management Group, Framingham, MA, 2002.

[10] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, Reading, MA, 1998.

[11] B. Henderson-Sellers, B. Unhelkar, *OPEN Modeling with UML*, Addison-Wesley, Harlow, England, 2000.

[12] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture Process and Organization for Business Success*, Addison-Wesley, Harlow, England, 1997.

[13] M. Jazayeri, A Ran, and F. van der Linden, *Software Architecture for Product Families: Principles and Practice*, Addison-Wesley, Boston, MA, 2000.

[14] C. Hofmeister, R. Nord, D. Soni, *Applied Software Architecture*, Addison-Wesley, Reading, MA, 2000.

[15] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*, Addison-Wesley, Reading, MA, 2000.

[16] M. Hammer, *Beyond Reengineering: How the Process-Centered Organization is Changing Our Work and Our Lives*, Harper Business, 1996.

[17] K. Preiss, S.L. Goldman, R.N. Nagel, *Cooperate to Compete: Building Agile Business Relationships*, Van Nostrand Reinhold, New York, 1996.

[18] P. Seng, A. Kleiner, *et al.*, *The Dance of Change: The Challenges of Sustaining Momentum in Learning Organizations*, Doubleday/Currency, New York, 1999.

[19] *OMG Unified Modeling Language Specification*, v1.4, Object Management Group, Framingham, MA, 2002.

[20] *The Common Object Request Broker: Architecture and Specification*, v3.0, Object Management Group, Framingham, MA, 2002.

[21] *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std 1471-2000, Institute of Electrical and Electronics Engineers, New York, 2000.