

Eclipse Process Framework Composer:

Part 2: Authoring method content and processes

Second Revision, April 2007

by Peter Haumer, phaumer@us.ibm.com
Solution architect, IBM Rational Software

Part Two of this introduction to the Eclipse Process Framework Composer (EPF Composer) details the concepts that define method content and processes in EPF, and explains cataloging method content and defining processes, with examples.

Keywords: Eclipse Process Framework, process engineering, project management, software development process, requirements management, configuration management, agile development, software development best practices, IBM Rational Method Composer, Rational Unified Process.



Author bio: Dr. Peter Haumer is an Eclipse Process Framework committer. He is solution architect for the IBM Rational Method Composer product platform and responsible for defining next generation process engineering tools. He represents IBM at the OMG in the SPEM 2.0 initiative.

In Part One of this two-part introduction to EPF Composer I discussed key concepts and typical usage scenarios for the tool. In Part Two, I will provide more details about the concepts that define method content and processes in EPF. I will start by reviewing the three main elements for building method content -- namely, roles, tasks, and work products -- and their relationships. I will then discuss guidance and the different kinds of guidance supported by EPF Composer. I will show you how to organize catalogs of your method content using standard and custom categories. I will then move on to defining processes in EPF Composer providing you with a rationale for EPF's process representation and walking you through a set of examples for the different kind of processes you can use and create in EPF Composer.

The paper presents the EPF Composer version 1.0.2 that is available as a free download under the Eclipse Public license at the EPF homepage¹⁴. IBM also offers a commercial version of the EPF Composer tool named IBM Rational Method Composer¹⁵ (RMC) that ships with the IBM Rational Unified Process (RUP) and provides additional enterprise capabilities not described in this article.

[Note: The number of sections (beginning with Section 4) in this article, figures (beginning with Figure 9) and footnotes (beginning with footnote number 17) is an intentional continuation from Part One.]

¹⁴ <http://www.eclipse.org/epf/>

¹⁵ <http://www-128.ibm.com/developerworks/rational/products/rup>

4 Managing intellectual capital and assets as method content and guidance

Let's consider the three main elements for building method content -- roles, tasks, and work products -- and their relationships.

4.1 Roles, task, work products

One of the most commonly quoted phrases in software engineering literature is “software development is a team sport,”¹⁶ indicating that software is (still) created by human developers who perform development tasks in which they heavily collaborate and exchange the products of their work with each other. In that sense, the Eclipse Process Framework applies the very common and even a standardized approach¹⁷ to define development methods, as depicted in Figure 9, by identifying development roles that represent a set of related skills, competencies, and responsibilities of a development team. These roles are responsible for specific types of work products: for example, developers are responsible for source code or system analysts are responsible for use case specifications. For creating and modifying work products, roles are assigned to perform tasks that have specific types of work products as input and output.

¹⁶ See, for example, Walker Royce, *Software Project Management: A Unified Framework*, Addison Wesley Professional, 1998.

¹⁷ OMG, “Software Process Engineering Metamodel,” version 2.0, Final Adopted Specification, ptc/07-03-03, <http://www.omg.org/cgi-bin/doc?ptc/07-03-03>.

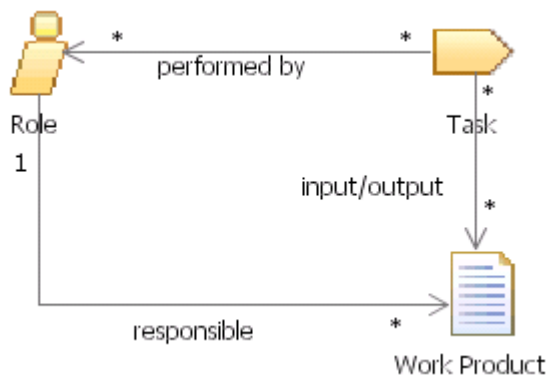


Figure 9: Simplified conceptual model of the core method content concepts of role, task, and work product.

You can assign a task to more than one role in EPF, which emphasizes the collaborative nature of software engineering, because many tasks require that people with different skills, background, and stakes work together. For example, a task called “prioritize requirements” needs to combine skills and expertise from different roles. You need the system analyst’s skills to represent the end user’s needs and priorities. You need the architect’s skills to assess feasibility and costs of features for the project manager to be able to define the cost/benefit ratio. You also need the project manager’s skills to assess how many development resources and skills are available to be able to allocate and prioritize requirements realizations to different iterations. Therefore, prioritizing requirements requires the collaboration of individuals playing roles with all these different skills. In EPF Composer, you assign all the roles you need to tasks to clearly define what skills are required to do the work. When actually performing this work in a project, a project manager can still decide whether to assign one person per role or to have them perform several roles at the same time.

The concept work product defines an abstract generalization of three concrete types of work products used in EPF: artifacts, deliverables, and outcomes. Artifact is a special kind of work product that provides a description and definition for tangible, non-trivial work products. Artifacts may be composed of other artifacts. For example, a model artifact can be composed of model elements, which are also artifacts. Artifacts are work products for which it is typically possible to define examples and templates for. They may also serve as a basis for defining reusable assets. Deliverables are used for packaging other

work products for delivery to an internal or external party. They define typical or recommended content in the form of work product selections and deliverable-specific documentation that would be packaged for delivery. Outcomes are intangible work products that are a result or state, such as an installed server or optimized network. A key difference between outcomes and artifacts is that outcomes are not candidates for harvesting as reusable assets.

4.2 Mapping method description into method content

In Part One of this article, I described method content as a way to represent fundamental development methods and techniques harvested from software engineering literature and represented in EPF Composer with the concepts of role, task, and work product. The Rational Unified Process, for example, which is available with a commercial variant of EPF Composer, called Rational Method Composer, contains a full range of development methods for most key development disciplines. Some examples are

Use-case analysis. A number of books¹⁸ on this subject describe how a designer systematically creates object-oriented analysis models from use case specifications.

Developing and scoping a vision. Other authors¹⁹ have defined a method for mapping and communicating stakeholder problems to business needs to system features (requirements) used for project scoping.

Testing. Cem Caner et al. have defined and published various methods²⁰ for systematically planning for tests, identifying and designing test ideas, and doing systematic regression testing.

¹⁸ See I. Jacobson et al., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992; Peter Eeles, Kelli Houston, and Wojtek Kozaczynski, *Building J2EE Applications with the Rational Unified Process*, Addison Wesley, 2002; and my own chapter titled “Use Case-Based Software Development”, in *Scenarios, Stories, Use Case*, edited by Ian Alexander and Neil Maiden, Wiley, 2004.

¹⁹ Dean Leffingwell and Don Widrig, *Managing Software Requirements: A Use Case Approach*, Second Edition, Addison-Wesley, 2003; also Kurt Bittner and Ian Spence, *Use Case Modeling*, Addison-Wesley, 2003.

²⁰ Cem Kaner, Jack Falk, and Hung Quoc Nguyen, *Testing Computer Software*, 2nd Edition. John Wiley & Sons, 1999.

Methods like these have been mapped by the RUP Content Team from their sources into one or more task. As shown in Figure 10, every task is organized into a series of steps, associated with performing roles and input/output work products, and with guidance providing additional background information and detail.

Task: Use-Case Analysis

This task describes how to develop an Analysis-level Use-Case Realization from a Use-Case.

Discipline: [Analysis & Design](#)

[Expand All Sections](#) [Collapse All Sections](#)

Purpose

- To identify the classes which perform a use case's flow of events
- To distribute the use case behavior to those classes, using analysis use-case realizations
- To identify the responsibilities, attributes and associations of the classes
- To note the usage of architectural mechanisms

[Back to top](#)

Relationships

Roles	Primary Performer: <ul style="list-style-type: none"> • Designer 	Additional Performers:
Inputs	Mandatory: <ul style="list-style-type: none"> • Use Case 	Optional: <ul style="list-style-type: none"> • Use-Case Realization • Analysis Class • Analysis Model • Design Model • Use-Case Model • Supplementary Specifications • Software Architecture Document • Glossary • Project-Specific Guidelines
Outputs	<ul style="list-style-type: none"> • Use-Case Realization • Analysis Class • Analysis Model 	

[Back to top](#)

Steps

[Expand All Steps](#) [Collapse All Steps](#)

- [Create Analysis Use-Case Realization](#)
- [Supplement the Use-Case Description](#)
- [Find Analysis Classes from Use-Case Behavior](#)
- [Distribute Behavior to Analysis Classes](#)
- [Describe Responsibilities](#)
- [Describe Attributes and Associations](#)
- [Reconcile the Analysis Use-Case Realizations](#)
- [Qualify Analysis Mechanisms](#)
- [Establish Traceability](#)
- [Review the Results](#)

[Back to top](#)

More Information

Checklists	<ul style="list-style-type: none"> • Use-Case Realization
Guidelines	<ul style="list-style-type: none"> • Use-Case-Analysis Workshop • Use-Case Realization • Sequence Diagram • Communication Diagram • Analysis Class
Tool Mentors	<ul style="list-style-type: none"> • Capturing the Results of Use-Case Analysis Using Rational Rose • Creating Use-Case Realizations Using Rational Rose • Performing Use-Case Analysis Using Rational XDE Developer • Managing Collaboration Diagrams Using Rational Rose • Managing Sequence Diagrams Using Rational Rose • Managing the Design Model Using Rational Rose • Performing Use-Case Analysis Using Rational Software Architect

[Back to top](#)

Figure 10: The method of performing Use Case Analysis represented as a task in EPF Composer. The example shows how a task is rendered into HTML and presented to a user. Detailed step descriptions are provided in collapsible sections. All blue text in this figure represents hyperlinks to pages of the related method content elements.

For example, Figure 10 depicts how the Use Case Analysis method is represented in RUP as a task performed by the role designer. The task has the work product use case as a mandatory input. It has several other work products as optional inputs, such as use-case realization, analysis class, glossary, and so on. As output work products, it defines a use-case realization as well as the analysis model comprising of analysis classes. The task is linked to various guidance elements such as a check list, guidelines for specific techniques such as performing a use case-analysis workshop, guidelines discussing the details of work products such as use-case realizations, analysis classes, and so on. Finally, there are tool mentors that describe how to perform use case analysis using modeling tools.

The task itself is structured into steps, although these steps do not have to be performed in a strict sequence. In fact several of these steps might only be performed in particular situations and not for each and every use case. For example, the step “Reconcile the Analysis Use-Case Realizations” requires that you already performed this task more than once so that you have several use-case realizations to reconcile. Hence, steps describe units of work within the task that the task performer can choose from when doing the work. As you will see in the “Managing Process with EPF Composer” section, it is the process that actually selects which steps shall be performed at what point of time. The task with its steps defines the method of use case analysis with all of its variants of performing it, describing all the work that gets done throughout a development lifecycle that contributes to the achievement of the task’s purposes. It is the application of the method element in a process that provides it with context that allows you to decide on what parts of the method you are actually performing at what point in the lifecycle.

Defining and using tasks in EPF Composer intentionally has a lot of commonality with how use cases are defined and used in software engineering. As a use case, a task shall provide a clear purpose in which the performing roles achieve a well defined goal that provides value to a stakeholder. An individual step of a task, such as “create analysis use case realization” in our example above, does not provide a value on its own. The use case realization concept by itself only represents the container for the analysis elements

identified in the analysis. It is therefore a means to the end of achieving the purposes of the analysis task as listed at the top of Figure 10.

Defining tasks with a clear purpose or goals provides similar advantages for identifying and managing tasks as it does for use cases. Stakeholders such as process implementers or developers can more easily relate to their tasks because they see the larger context of their purpose: why they are doing this work. Defining and assigning too many low level units of work to people is hard to manage and hard to communicate. Also communication and education of tasks becomes much easier if they fulfill a level of granularity that relates to clear development goals. As a result, performing a task can range from a few hours to many days. Hence, process estimation cannot be performed by counting tasks, but need to be performed by considering estimates for every task individually. (However, it is recommended that you define estimates for occurrences of tasks in processes, which already provide more specific context as described above.)

Another rule for a task's granularity is that it usually affects one or only a small number of work products. In other words, a task shall not have too many different output work product types. In such a case, it might either be that you defined too fine-grained work products that should be combined to address a particular purpose. It could also be that your tasks perform work of several development methods which should be refactored into separate ones to increase reusability of the task and focus on specific goals.

As with use cases, the aim is to provide complete definitions of doing all the work that needs to be done to achieve the task's goal. This includes alternative as well as optional steps. In this sense, the task's steps map to use case flows that can be placed in many different combinations in a process to derive different "scenarios," in this case development scenarios. Assigning input and output work product relationships to a task is illustrated in Figure 11.

To complete the use case analogy, please note that the performing roles of a task do not map to use case actors because actors are external users. Roles represent the developers performing the actual work.

Therefore, roles are similar to control objects in analysis use-case realization or business workers in business use-case realizations.

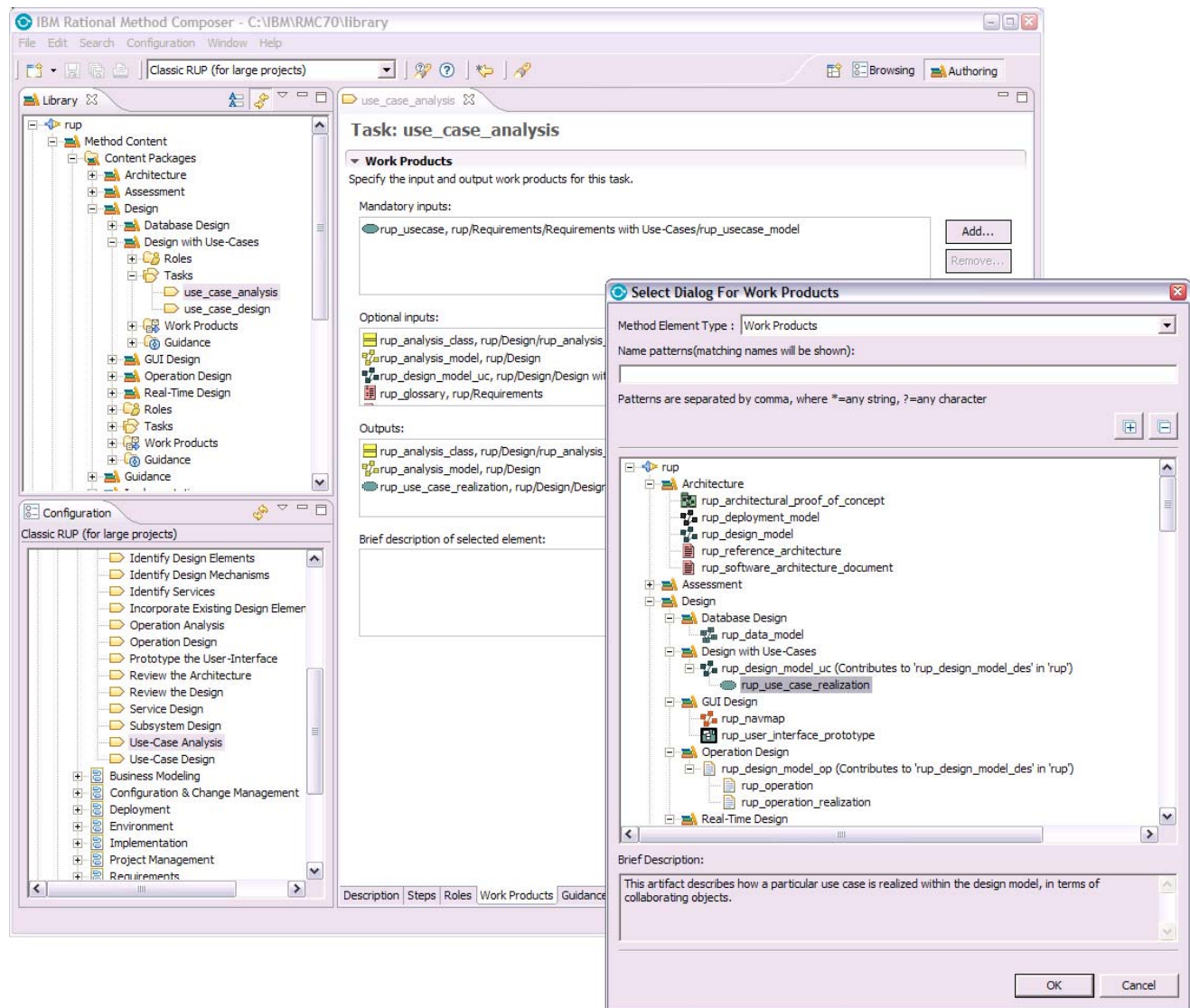


Figure 11: Assigning input and output work product relationships to a task in an EPF Composer task editor using simple add and remove buttons and dialogs.

4.3 Guidance and categories

Guidance elements let you add any additional information that makes your method content more complete and allows you to factor details into separate descriptions, as shown in Figure 12. The concrete guidance

kinds supported in EPF Composer cover the addition of whitepapers, checklist, templates, or term definitions. See Table 1 below for a complete list of supported guidance kinds.

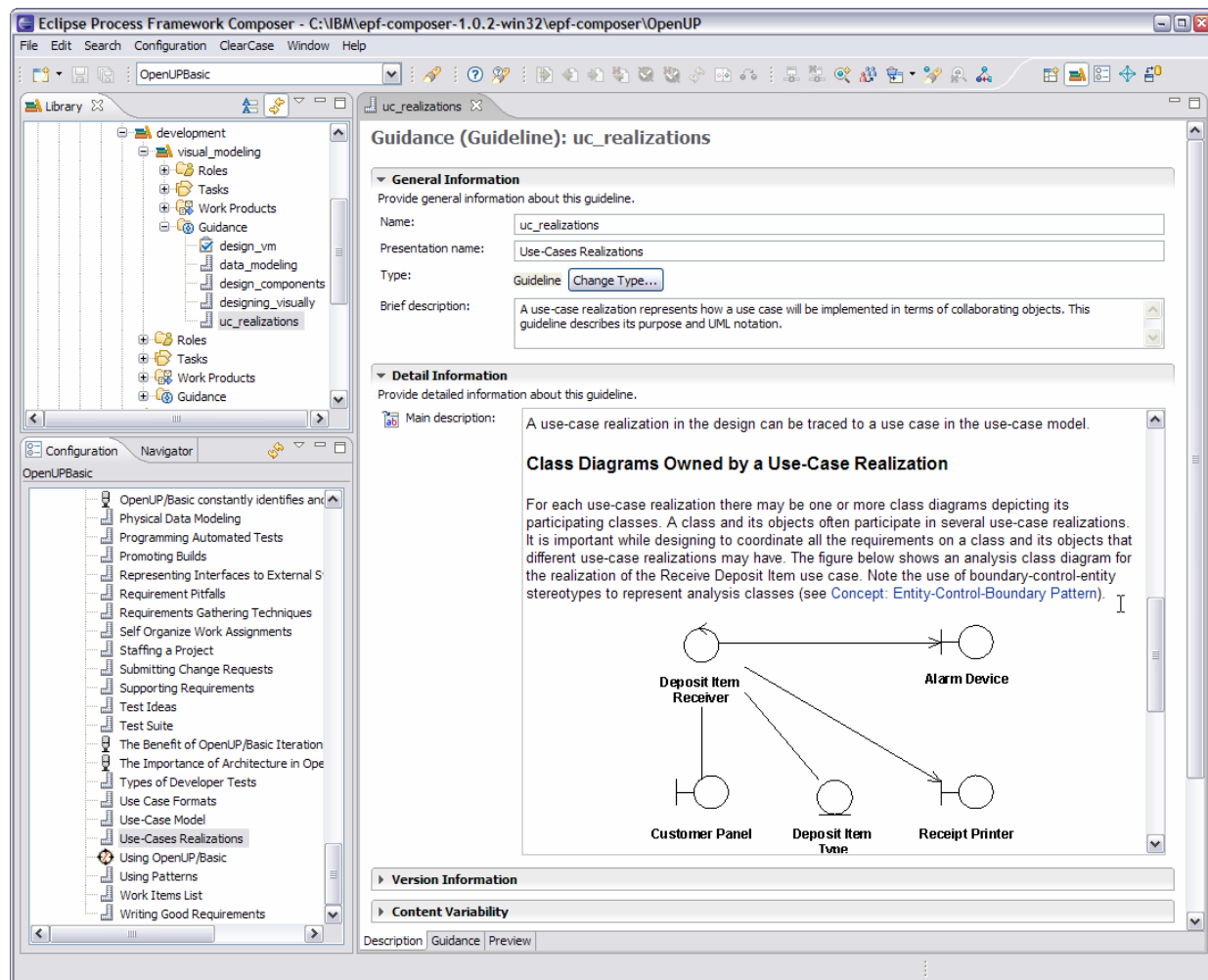


Figure 12: Guidance is authored in EPF Composer with specific form-based editors. This figure shows the editor for the Guideline kind. It basically consists of a free-form text field that allows editing whitepaper-like documentation. Other guidance kinds provide more specific editors; for example, the Checklist kind is edited with a check item editor.

EPF Composer supports you in authoring such guidance elements using form-based editors as depicted in Figure 12. You can then relate your guidance to your roles, tasks, or work products using simple dialogs for either creating links inline to your documentation (for example, inside a step description, a work product definition, or the text of another guidance element) or in the separate reference sections such as the “More Information” section that you saw in the example of Figure 10.

Table 1: A list of the different guidance kinds supported by EPF Composer. Different guidance kinds can be related to different elements. For example a template is typically associated only with work products and not tasks or roles.

Checklist	Identifies a series of items that need to be completed or verified. Checklists are often used in reviews such as walkthroughs or inspections.
Concept	Outlines key ideas associated with basic principles underlying the referenced item. Concepts normally address more general topics than guidelines and span several work product and/or tasks or activities.
Example	Provides an example of a completed work product or performed tasks and activities.
Guideline	Provides additional detail on how to perform a particular task or grouping of tasks, or that provides additional detail, rules, and recommendations on work products and their properties.
Practice	Represents a proven way or strategy of doing work to achieve a goal that has a positive impact on work product or process quality. Practices are defined orthogonal to methods and processes. They could summarize aspects that impact many different parts of a method or specific process.
Report	A template for an automatically generated description with extracted content one or several work products.
Reusable Asset	Links a prepackaged solution to a problem for a given context. Examples of asset are design patterns or mechanisms, solution frameworks, etc. Asset packaging and consumption is directly supported by IBM Rational's design and construction tools.
Roadmap	A linear walkthrough of a complex process or activity. (This is process specific guidance.)
Supporting Material	A catch-all for other types of guidance not specifically defined elsewhere. It can

	be related to all kinds of content elements.
Template	For a work product, provides a predefined table of contents, sections, packages, and/or headings, a standardized format, as well as descriptions on how the sections and packages are supposed to be used and completed.
Term Definition	Defines terminology and is used to build up the Glossary.
Tool Mentor	Shows how to use a specific tool to accomplish some piece of work, either in the context of, or independent from, a task or activity.
Whitepaper	Similar to concept, but has been externally reviewed or published and can be read and understood in isolation of other content elements and guidance.

In addition to providing supplementary informal content with guidance, you can use categories to organize and index your content presentation, as shown in Figure 13. EPF Composer provides a set of predefined standard categories for its method content elements such as disciplines categorizing tasks, domains categorizing work products, role sets grouping related roles and so on. EPF Composer also allows you to define your own custom categories that you can use to define your own logical organization and presentation structures for roles, tasks, work products, as well as guidance. All the tree browser views you see in the published EPF Composer content are generated based on these categories. Categories themselves are considered content elements (yes, you can also categorize categories themselves), because they can also contain detailed rich text descriptions defining and summarizing the category. Moreover, you can also use custom categories to create your own indexes and catalogs on your content. For example, you can define categories for CMMI²¹ levels or practices and categorize content such as your tasks based on these levels.

²¹ Capability Maturity Model Integration. See the Software Engineering Institute Website for more information, at <http://www.sei.cmu.edu/>

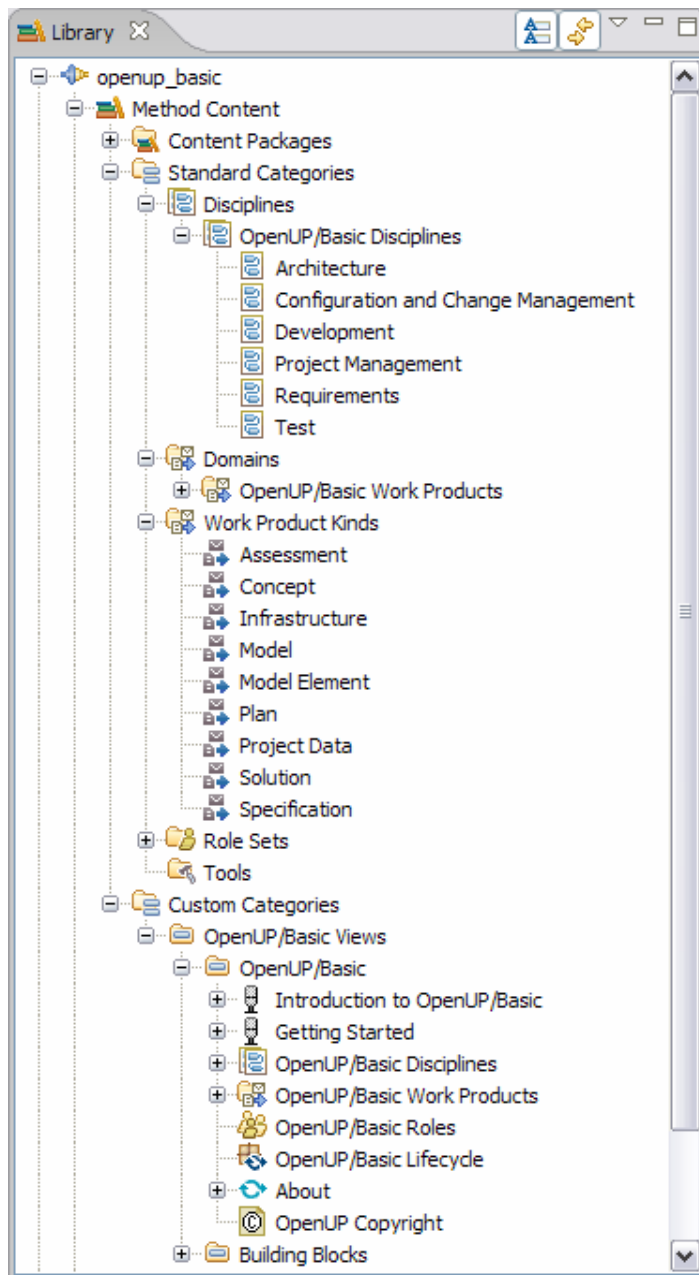


Figure 13: Examples for standard and custom categories defined for OpenUP/Basic. Disciplines and Work Product Kinds are examples for standard categories. Custom Categories can be used for classifying your content as well as defining navigation structures for the published Web site.

5 Managing processes with Eclipse Process Framework Composer

In Part One of this article, I provided you with a general overview of processes in EPF. In this section, we will look at the details of creating processes. I will discuss how EPF Composer facilitates your process tailoring work with reusable building blocks expressed as capability patterns and method content. I will show you how to create agile processes with implicit method content, as well as enriched processes that refer to concrete method content. In particular, we will look at how relationships defined for method content can guide you to systematically completing your processes.

5.1 A rationale for EPF's process representation and presentations

Development processes define how development projects shall be executed. One of the most common characteristics found within the many different definitions of process in literature is sequencing of phases and milestones expressing a lifecycle of the product under development. Processes also define how to get from one milestone to the next by defining sequences of work, operations, or events that usually take up time, expertise, or other resource and which produces some outcome.

Reference frameworks for development processes such as CMMI define different levels of maturity for processes. Each level entails different characteristics for the process definition as well as enactment in an organization or project for each level. For example, CMMI defines a “managed process” as performed activities that can be recognized as implementations of development practices. Such a process has certain characteristics: it is planned and executed in accordance with policy; it employs skilled people having adequate resources to produce controlled outputs; it involves relevant stakeholders; it is monitored, controlled, and reviewed; and it is evaluated for adherence to its process description. By contrast, a “defined process” is a managed process that is tailored from the organization’s set of standard processes according to the organization’s tailoring guidelines. A defined process has a maintained process description and contributes work products, measures, and other process-improvement information to the organizational

process assets. As you will see throughout this section, RMC processes support many of these characteristics.

As we saw in the previous section, method content is defined as isolated content. It treats tasks as a kind of use case for achieving specific development goals. For creating a process, you now need to define how such methods are applied and sequenced in a development lifecycle. Figure 14 depicts conceptually how method content -- defined by its three main elements (or dimensions) tasks, roles and work products -- is applied in a process along the timeline (the fourth dimension).

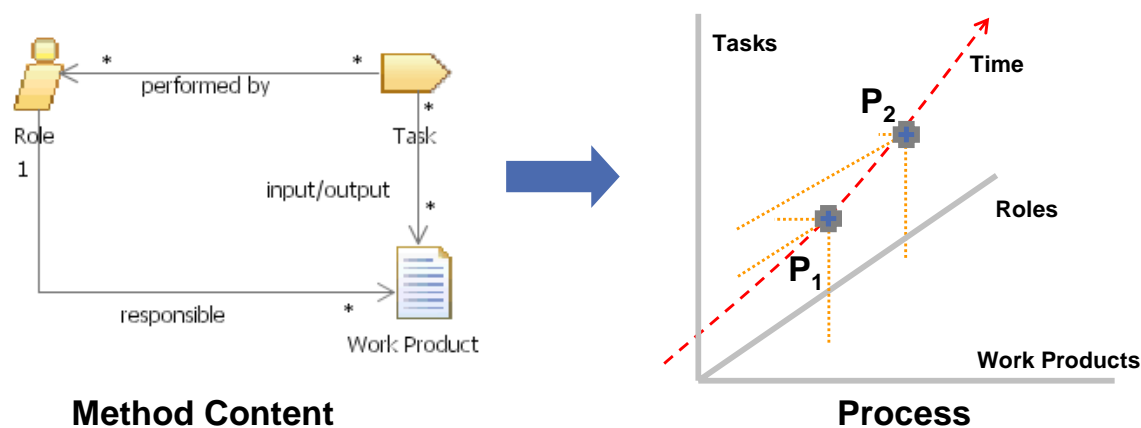


Figure 14: Applying method content in a process. The figure shows how method content defined with tasks, roles and work products is sequenced in processes.

Whereas traditional waterfall-based developed processes would be able to define more or less straight lines for the tasks to be performed, modern iterative development processes, such as OpenUP/Basic, RUP, Scrum, and XP, require more complex structures to represent increments of development. In these structures, work is packaged into repeatable chunks in which tasks are performed concurrently and repeatedly.

These varieties of development approaches also require different degrees of formality and levels of detail for their process descriptions. Traditional development processes define phases based on predefined deliverable types, providing exact descriptions of the sequences of work that produce these deliverables. Agile development processes only ask for informal work descriptions because they assume self-organizing teams will be able to perform the work without guidance and without control once the development goals have been defined. They also focus on scoping of iterations of work with less deliverable-

oriented but more capability-oriented (results-based) milestones. Finally, modern scalable iterative processes provide explicit descriptions of the work to be performed with a variable and tailorable amount of detail depending on the type of project and maturity of the organization. RMC aims to support all three of these approaches.

To support complex process structures as well as various degrees of formality, traditional iterative development processes are modeled using nested workflow representations. This allows you to define complex process structures as a hierarchy of nested diagrams in which each diagram node could be refined or detailed with a whole new diagram. For example, the RUP always defined its processes with UML-like activity diagrams. Activity diagrams can express sequencing using control flows combined with parallelism of work using fork, merge, and join constructs.

A second popular representation for traditional waterfall as well as iterative development processes is work-breakdown structures. Breakdown structures are similar to nested activity graphs because they also allow the refinement of work definitions via nesting. They provide an easy way to sequence work by using what is called predecessor dependencies, which have some similarity to control flows. All elements that are not related via these predecessors are free to be performed in parallel. Another advantage of the breakdown structure representations is that it is very popular in project enactment and management applications. Using breakdown structures in EPF Composer supports bridging the gap between process definition and process enactment in projects. EPF Composer integrates these two popular process representations of activity diagrams and breakdown structures into one format that it maintains internally to represent processes. Based on this format, RMC supports both presentations interchangeably, as shown in Figure 15.

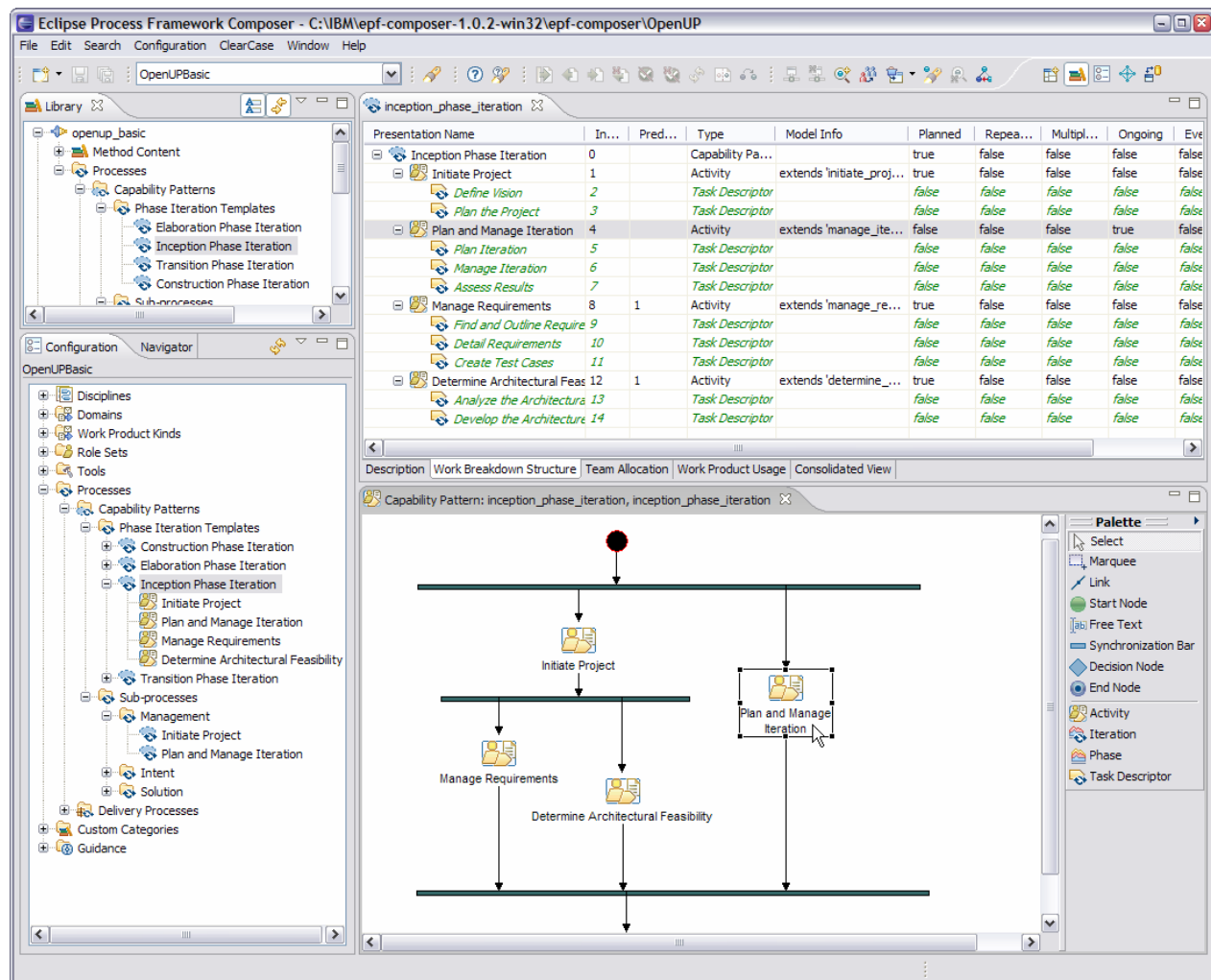


Figure 15: Two presentations for the same process structure. The process *Inception Phase Iteration* is consistently maintained in a breakdown structure as well as activity diagram view. Changes in one view are translated to the other view if possible. Presentation-specific elements, such as the synchronization bars in the activity diagram, are not presented in the breakdown structure, but appropriate predecessor dependency information is still derived from the diagram following simple mapping rules.

The example presented here depicts a process in both presentations for Inception phase iterations from the EPF's OpenUP/Basic.²² An EPF Composer process author can work on either of these two process presentations, and changes are automatically reflected by EPF Composer in the other view because both views are rendered from one underlying data structure. Of course, both presentations are not completely equivalent and contain information the other presentation does not contain, such as the synchronization bars in

the diagram of Figure 15 as well as decision points not available in breakdown structures. In this case, the other presentation view omits this information but provides a consistent mapping on it as we see with the predecessor dependencies shown in Figure 15. For example, drawing a control-flow link from one activity to another directly or via a synchronization bar creates a finish-to-start predecessor dependency between activities in the breakdown structure presentation. As you can see in the top of Figure 15, predecessors are listed according to the control flows in the diagram. For example, “Manage Requirements” and “Determine Architectural Feasibility” refer to the element with Index number 1 as its predecessor which is the activity “Initiate Project.”

5.2 Creating a Process with Breakdown Structures

Processes such as delivery processes or capability patterns (described in Part One of this article) are created in EPF Composer’s process editor as a breakdown of nested activities. Every activity in this breakdown can present its sub-activities with an activity diagram as we have seen in the last section.

Hence, “activity” is the main concept for defining processes. Many elements you see inside the process editor derive from the activity concept; in other words, they are specializations of activity. Processes themselves, such as capability patterns, are actually nothing else but special activities that allow processes to be nested inside activities. This provides a realization of the process pattern mechanisms we defined with capability patterns. Phases and iterations are also created as activities in EPF Composer’s process editor. The only concepts you find in EPF Composer’s processes that are not activities are descriptors (explained farther below) and milestones.

Creating a process with a lifecycle model in EPF Composer is very similar to creating a plan in a planning tool. Figure 16 depicts an example of how you begin developing an OpenUP-like process by defining phases, iterations, and milestones.

²² See Ricardo Balduino, Brian Lyons, “OpenUP/Basic - A Process for Small and Agile Projects”, at http://www.eclipse.org/epf/general/OpenUP_Basic.pdf, eclipse.org/epf, 2006.

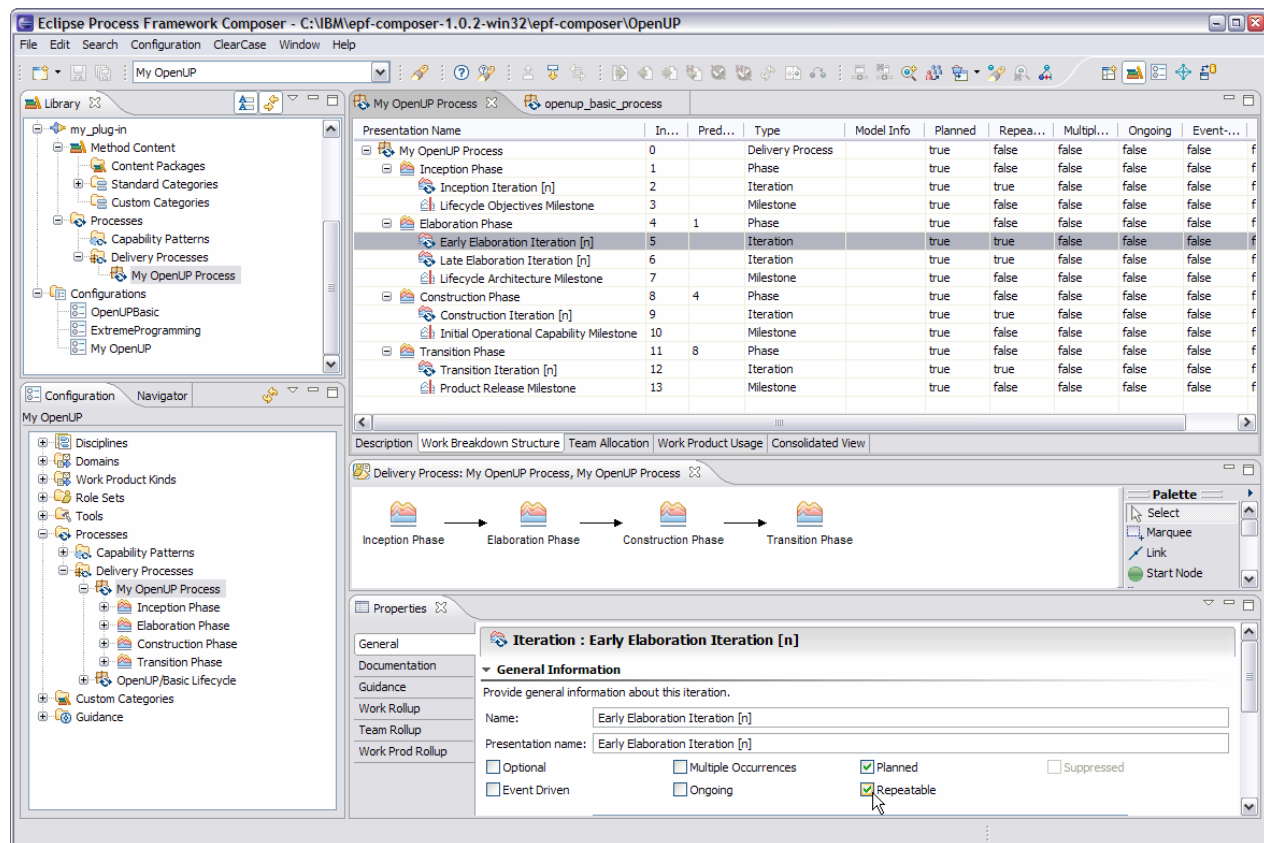
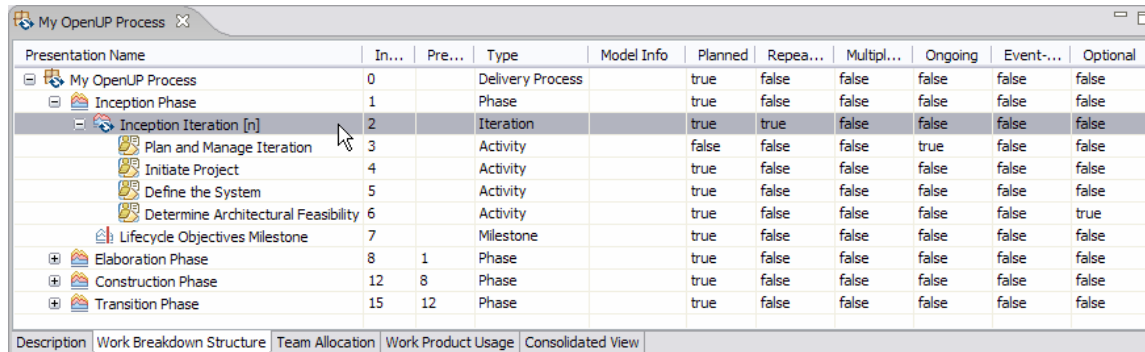


Figure 16: Creating a lifecycle model for an OpenUP-like process using the breakdown structure editor in EPF Composer. The screen capture also shows the activity diagram for the top-level activity named 'My OpenUP Process' as well as the properties window that allows you to document the activity currently selected in the breakdown editor.

As you have already seen in Figure 7 of Part One of this article, which showed a Scrum-like process lifecycle, you are not limited in EPF Composer to using the UP-like lifecycle model because EPF Composer supports you in the creation of almost any kind of lifecycle model.

The example process in Figure 16 defines four phases because it follows the UP lifecycle model. Its four phases have to be performed in sequence, which is expressed via the predecessor dependencies defined in the example. Within each phase of the delivery process, you can have sub-activities (or sub-processes) for iterations. Within the four phases you could now, for example, model the process for an early versus a late elaboration iteration if they are different from each other in the way work is performed or work products are being evolved. For instance in RUP, early elaboration work requires more overhead on establishing the work environment and the initial executable architecture than late elaboration work.

As mentioned above, except for the descriptors and milestones, all elements listed in this breakdown are activities, can have their own activity diagram, and can therefore be refined by more activities, as depicted in Figure 17. Figure 17 shows how the Inception Iteration has been refined with four activities that define the high-level work that has to be performed in these types of iterations.



Presentation Name	In...	Pre...	Type	Model Info	Planned	Repea...	Multipl...	Ongoing	Event...	Optional
My OpenUP Process	0		Delivery Process		true	false	false	false	false	false
Inception Phase	1		Phase		true	false	false	false	false	false
Inception Iteration [n]	2		Iteration		true	true	false	false	false	false
Plan and Manage Iteration	3		Activity		false	false	false	true	false	false
Initiate Project	4		Activity		true	false	false	false	false	false
Define the System	5		Activity		true	false	false	false	false	false
Determine Architectural Feasibility	6		Activity		true	false	false	false	false	true
Lifecycle Objectives Milestone	7		Milestone		true	false	false	false	false	false
Elaboration Phase	8	1	Phase		true	false	false	false	false	false
Construction Phase	12	8	Phase		true	false	false	false	false	false
Transition Phase	15	12	Phase		true	false	false	false	false	false

Figure 17: Detailed process showing how the Inception Iteration activity has been refined with activities that define the high-level work for this type of iteration.

Also note the information provided in the columns on the right. In addition to predecessor information, the process author who created these activities also specified that certain activities are to be considered as ongoing (in a plan derived from this process, these activities dynamically have the same duration as the parent activity), repeatable (they can be performed several times in sequence), or with multiple occurrences (they can be performed several times in parallel by different teams).

A process author working with agile self-organizing teams could now stop and consider this process of Figure 17 to be finished. It provides a complete development process by defining phases, different iteration kinds, milestones and the high-level activities to be performed. It might provide just the amount of detail and formality required for an agile project. However, in the following section we show you how you apply method content to your process. Such method content could define which work products shall be produced or define when which tasks should be performed as part of these activities. Such method references provide another level of detail in the breakdown that can be used for planning and executing

the process. It could also serve as guidance for the development team that does not want to map this level of detail to planned and tracked work in a project plan.

5.3 Refining processes with method content descriptors

An activity in EPF Composer can be refined with other nested activities, with references to method content called descriptors, or a combination of both. A descriptor is basically a reference object inside a process that, on one hand, represents the occurrence of a method content element such as a task or work product inside an activity, but on the other hand also has its own relationships and documentation attributes that define how this particular occurrence of the method content element might be different from its default definition.

For example, as we saw in Figure 5 of Part One of this article, a task can be applied to a process in EPF Composer by simply dragging it on top of an activity. However, instead of creating a copy of the task, it creates a task descriptor. This descriptor inherits properties and relationships from the task that can now be extended and overwritten. As a result, every activity defines its own local sets of descriptors. Every descriptor within such an activity only contains the relationships and information specific to the particular situation or context of the process in which it is applied. For example, the role “Project Manager” normally is responsible for a long list of different work products such as project plans, iterations plans, risk lists, work item lists, and so on. The method content element Project Manager role lists all of these relationships, providing the complete list of the work products a Project Manager would ever be responsible for. However, in the context of a specific activity, a role descriptor for a Project Manager would only list the work products the manager is responsible for in the context of this one activity at this point in the process. Refer to the upper process view of Figure 19 for an example. You see there that the Project Manager role is represented by a descriptor each in the activity Manage the Project as well as the activity Initiate the Project. In the context of the first activity, only the project manager’s responsibility to the “Iteration Plan” and “Work Items” list is relevant. The second activity does not deal with these artifacts at all and therefore only the responsibility to the “Project Plan” is represented for this role descriptor.

When adding descriptors to a process, you can start in any of the three dimensions depicted in the conceptual overview of Figure 14. EPF Composer provides you with three process views that allow you to work in a work-breakdown-centric, work product-centric, or role-centric fashion. Regardless of where you prefer to start, EPF Composer can support you in completing the information in the other two dimensions, EPF Composer uses interactive wizards that dynamically retrieve information from the method content based on the relationships illustrated on the left of Figure 14, and propose candidates for creating additional descriptors.

For example, Figure 18 shows a scenario in which the process author drags the “Project Manager” role into the activity “Plan and Manage the Iteration,” thus expressing that this role is performing work in this activity.

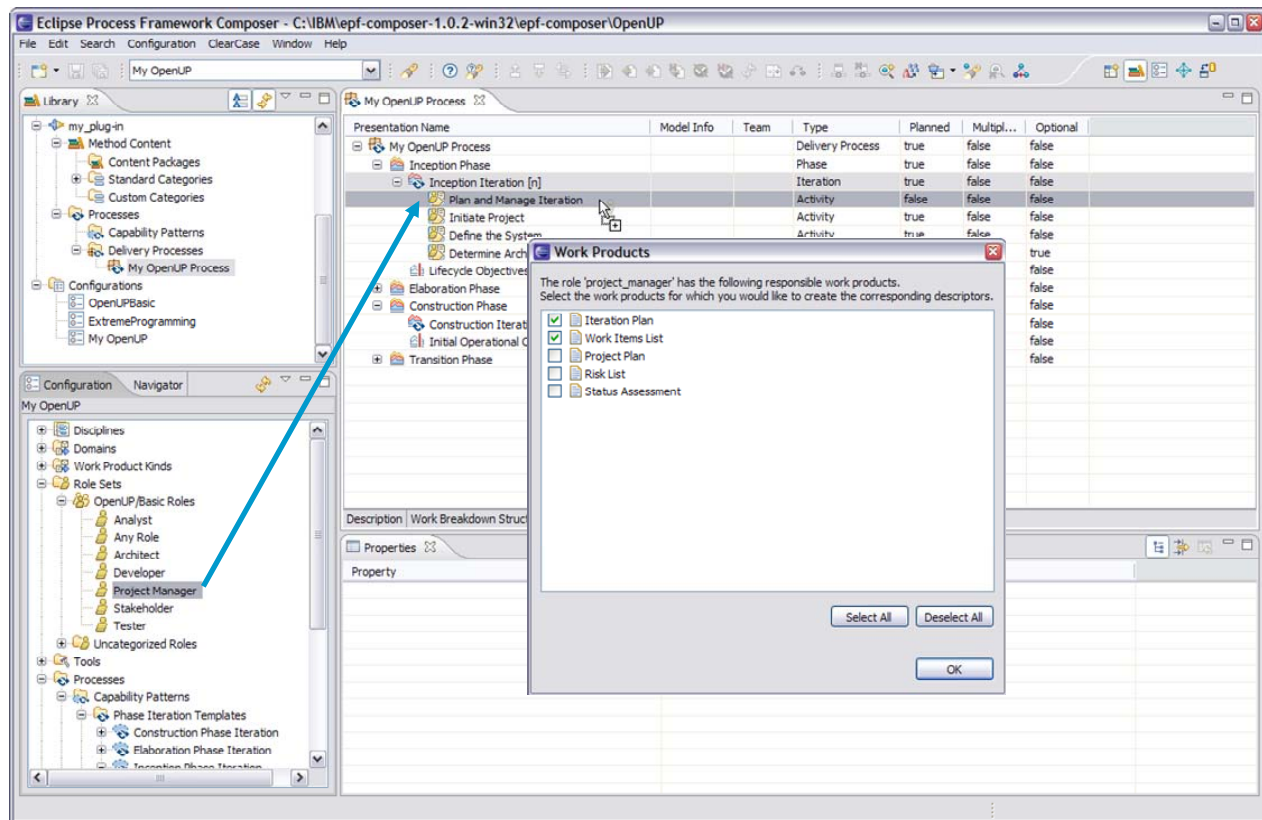


Figure 18: After applying the project manager role to an activity, EPF Composer asks the user if he wants to add to the process any of the work products this role is typically responsible for. It retrieved this information by checking relationships defined in the method content. After the user selects work products, EPF Composer adds them to the process as work product descriptors and establishes the responsibility relationship between them.

To guide the user in completing the activity based on the knowledge structures it has available in its method content, EPF Composer now proposes that the process author could further add any of the work products to the activity that the Project Manager is typically responsible for. In the case of Figure 18, the process author selected the two artifacts “Work Items List” and “Iteration Plan.” EPF Composer adds descriptors for these two work products to the same activity “Plan and Manage the Iteration” and makes the Project Manager descriptor the responsible role for these work products descriptors. After adding these new work product descriptors to the process, EPF Composer continues making suggestions (not visible in Figure 18). This time, it prompts all the tasks that list the selected work products as outputs because these tasks would be good candidates to add to the process as well.

Making these choices of adding descriptors in EPF Composer does not force you, however, to always create complete processes that contain task, work product, and role descriptors. You could also create more agile light-weight processes that, for example, only contain the work products to be worked on in activities, and optionally the roles responsible for these work products as depicted in the examples of Figure 19. This example shows in the lower view what we call entry and exit states for each work product descriptor within each activity.

The figure consists of two screenshots of the Eclipse Process Framework Composer interface, showing a process model for 'rup_process'.

Upper Screenshot: Consolidated View

Presentation Name	Index	Model Info	Type	P...	Repeat...	Ongoing
RUP-based Process	0		Delivery Pro...		false	false
Inception Phase	1		Phase		false	false
Inception Iteration [n]	2		Iteration		true	false
Manage the Iteration	3		Activity		false	true
Project Manager			Role Descriptor			
Iteration Plan		Responsible For	Artifact			
Work Items List		Responsible For	Artifact			
Initiate Project	4		Activity		false	false
Project Manager			Role Descriptor			
Project Plan		Responsible For	Artifact			
Analyst			Role Descriptor			
Vision		Responsible For	Artifact			
Define the System	5		Activity	4	true	false
Analyst			Role Descriptor			
Vision		Responsible For	Artifact			
Use-Case Model			Artifact			
Determine Architectural Feasibility	6		Activity	4	true	false
Architect			Role Descriptor			
Architecture		Responsible For	Artifact			
Lifecycle Objectives Milestone	7		Milestone		false	false
Elaboration Phase	8		Phase	1	false	false
Construction Phase	13		Phase	8	false	false
Transition Phase	16		Phase	13	false	false

Lower Screenshot: Work Product Usage View

Presentation Name	Type	Entry State	Exit State	Optional
RUP-based Process	Delivery Pro...			false
Inception Phase	Phase			false
Inception Iteration [n]	Iteration			false
Manage the Iteration	Activity			false
Iteration Plan	Artifact	defined	refined	true
Work Items List	Artifact	defined	refined	false
Initiate Project	Activity			false
Project Plan	Artifact	undefined	goals, milestones defined	false
Vision	Artifact	undefined	problem, needs defined	false
Define the System	Activity			false
Vision	Artifact	problem, needs defined	initial scope defined	false
Use-Case Model	Artifact	undefined	briefly described	true
Determine Architectural Feasibility	Activity			true
Architecture	Artifact	undefined	candidates id'ed	true
Lifecycle Objectives Milestone	Milestone			false
Elaboration Phase	Phase			false
Construction Phase	Phase			false
Transition Phase	Phase			false

Figure 19: A light-weight process without any task descriptors. The figure shows two views on the same process. The upper Consolidated View shows the breakdown defined for this process. Every activity defines performing roles as well as work products the roles are responsible for. The lower Work Product Usage view shows the process' work products defining for every work product descriptor the state at the beginning (entry) and end (exit) of every activity. In this process, the associated role is responsible for transforming its work products from the entry to the exit states.

Entry states express the state work products should be in before work on the activity can begin as well as exit states define the state the work product should be in before the activity can be assumed to be completed. In processes like these, roles can decide on their own methods for achieving the requested work product state, without the need to follow formal and prescriptive task descriptions. However, processes like these still contain enough formality to define which roles are in charge of which work products and what are the expected results.

6 Outlook

In this two part article I provided you with an overview of EPF Composer. I presented a high-level summary of its key concepts and capabilities, and we iteratively refined these concepts with detailed conceptual and tool-specific descriptions.

To get started with the EPF Composer tool itself, we recommend that you explore the tool's online help which contains several interactive tutorials that provide you with step by step instruction on how to perform the scenarios discussed in this paper.

Additional applications and capabilities of EPF Composer that were outside the scope of this initial article are provided in other articles and video recordings available on the www.eclipse.org/epf Web site.

7 Acknowledgements

None of the work presented in this paper would have become a reality without the dedication and passion of several remarkable teams. I would like to thank all EPF committer and contributors, the RMC/RUP content, tool, quality, and product development teams, the IBM Global Services Method team, the UMA steering committee with members from the Rational field and other IBM method teams, the Tivoli ITUP team, the OMG SPEM 2.0 team, as well as all the other early adaptors and beta-users, and last but not least, Rational ISSR management for making RMC and the EPF donation happen.