Improvements to the OPEN Process Metamodel

D. Firesmith and B. Henderson-Sellers

Over the last few columns in ROAD (refs. 1–3), we have been discussing various aspects of object-oriented processes and object-oriented process metamodels. This month, we describe some improvements to the metamodel of the OPEN process framework.

An Object-Oriented Process Metamodel

A process metamodel uses metamodelling techniques, such as those embodied in a modelling language like UML or OML, to describe not models but the underpinning process of developing work products (documentation, code) including those models. A third generation process should go further and describe the full lifecycle — *all* aspects of software development including people issues, organizational culture, tools and available technology (ref. 1). All these elements also need to be included in the metamodel.

The OPEN Process Metamodel: Version 1

The Software Engineering Process described by the OPEN methodological framework (ref. 4) is shown in **Figure 1**. This describes the computer side (and not the business side) of software development. This will be the focus of this paper although not the total focus of OPEN which also includes business issues (e.g. refs. 4, 5). The process is a combination of people and organizational culture; tools and available technology and the methodology which itself consists of techniques together with an iterative modelling process, producing work products documented using a modelling language.

In OPEN Version 1, the MODELLING PROCESS metaclass was then expanded into a number of Activities and Sequencing Rules. In turn Activities were seen to be an aggregation of tasks (**Figure 2**). Activities were linked to Work Products which were (incorrectly it turns out) called Deliverables which are created at least in part by the application of one or more techniques within this process framework.

The OPEN Process Metamodel: Version 2

The focus of this paper is to present improvements to the (Version 1) OPEN process metamodel described in the previous section. The motivation for undertaking this was the realization that the Version 1 metamodel skimped on its inclusion of people and organizations and, while including them, was deficient on the practical aspects of deliverables and work products such as the software and its associated documentation. At the same time, as we introduce more fully these various, hitherto-neglected metaclasses, we are cautious not to have the OPEN metamodel force a document-driven approach. OPEN, as a methodological process for developing OO software (ref. 4) needs to be remain flexible enough to be used in projects which are predominantly architecture driven, use case driven, responsibility driven, data driven, timeline driven, document driven or any other flavour of OO development.

The changes to be made are shown in Figures 3–10 and are based, in part, on suggestions in ref. 6. Three immediate name changes are seen: the sequencing is no longer carried out using a SEQUENCING RULES metaclass but with the ASSERTION metaclass (**Figure 3**), thus bringing it more in line with the notions of a contract-driven lifecycle, as used in OPEN. Secondly, it was recognized that the word "deliverable" could be interpreted

as meaning "delivered to end user/customer" whereas the word was used to include all work products, whether externally delivered or just needed for internal consumption of the software development team. Accordingly, in the summary Version 2 metamodel of Figure 3 (compared to the original version shown in Figure 2) we rename this metaclass as WORK PRODUCT — also a term in general use in other methodological approaches. Thirdly, the name MODELLING PROCESS becomes DEVELOPMENT PROCESS to reflect its scope as encompassing all the "software technology" aspects of building a system. As we can see, an important part of such a metamodel is a clear understanding of what all the terminology means i.e. the semantics. Definitions of the terms used in this and subsequent metamodel diagrams are given in **Table 1**.

So what else is new — and why are we introducing it? Well, a lot more detail about work products and the people and tools that produce them. **Figure 4** shows the "big picture". When software is being created, it may be within a single Project or as part of a larger, cross-project Programme (two kinds of Endeavor). Producers may be direct (people, rôles or tools) or indirect (teams, endeavors). The people involved play rôles within the team supporting the Endeavor. Producers perform Tasks, implemented by Techniques (as before) to generate Work Product Versions. Borrowing from ref. 8, we make explicit the fact that information needs to be kept on the evolution of Work Products by attaching a State Machine to each Work Product Version. Activities, which comprise a Development Cycle, are heterogeneous collections of Tasks; together, they are described by Work Units, sequenced by Assertions. By choosing the appropriate assertions, process engineers can obtain any well-defined development cycle from the traditional waterfall to a fully incremental, iterative, and parallel (IIP) development cycle.

A Work Product is then an aggregation of Work Product Versions, all of which are described using a Natural Language (e.g. English), a Modelling Language (e.g. UML, OML) and/or a Programming Language (e.g. Java, Smalltalk, Eiffel, C++). Furthermore, in an iterative and incremental lifecycle, as in OPEN, work product versions are created and/or updated at each Increment. This is either a Build which may be a Release to the customer or an internal build; or it may be what is known as a Drop: an internal release of either code or documentation. These increments are then fed back as input to the next iterative increment. The Increment and Work Product portion of Figure 4, when amplified (**Figure 5**), is more comprehensive, including as it does other types of work product including metrics, responsibility cards, requirements etc.

There are more people-oriented metaclasses in Figure 4 (as compared to Figure 2). As before a Programme is broken down into Projects. Each such Endeavour is staffed by an overall Team, typically decomposed into collaborating subteams. Both Teams and Endeavours can be considered to be kinds of Indirect Producers which are specialized kinds of Producers (as are Direct Producers). People play a Rôle in the team and, again, both Rôles and Persons may be considered as Direct Producers i.e. those people who *actually* create the work products. Another kind of direct producer is a Tool because tools also generate output directly. All of these kinds of direct producers perform one or more Tasks; in turn linked to Techniques as before. A more detailed view of the producer inheritance hierarchy is given in **Figure 6**. Both Programmes and Projects are classified as kinds of Endeavour. Teams and Endeavours are both indirectly responsible for the production of work products. Direct production is effected by tools, people or rôles (within teams). Typical tools are CASE (Computer Assisted Software Engineering) tools which may be

focussed at the code level (lower CASE) or at the model level (upper CASE) or tools that support natural language documentation. They thus support Programming Languages, Modelling Languages and Natural Languages, respectively — each a kind of Language (see also Figure 4). Team structures are highlighted in **Figure 7** which stresses the need for multiple sub-teams making up the overall project team. A team is an aggregation of rôles which collaborate to perform tasks needed in the generation of work products. Teams need to be cohesive, contain all required rôles, have high internal coupling/collaboration yet low external coupling, with a well-specified interface. Teams are characterized by their (a) definition, (b) responsibilities and (c) membership. Firesmith (ref. 9) lists 29 predefined types of team that may be staffed with 37 predefined rôles. For example, six of these (Marketing Team, Project Initiation Team, System Development Team, Software Development Team, Hardware Development Team and Support Team) form the project team, one of which (the software development team and its components) is shown in full detail in Figure 7. Reuse is catered for by the inclusion of a Reuse Team, not part of the Project Team *per se*, but interacting directly with it as twin components of the overall Development Organization.

The new way in which the sequencing of events in the process is scheduled is shown in more detail in **Figure 8** in which the rôle of Assertions is demonstrated in the context of Activities, Tasks and Work Products.

A metamodel should provide not merely a static architecture of predefined types and their relationships. By containing collaborating classes, the metamodel should allow process engineers to specify the behavior of classes and how they collaborate. For example,

Figure 9 is a simplified sequence diagram (looping and branching are ignored) that illustrates how metamodel components can collaborate to produce and baseline a version of a software requirements specification. Note that the goal is to model the reality of how the project teams and their members are to collaborate to produce work products.

Another metaclass introduced in this diagram (see also Figure 3) is that of Workflow, which is a *sequential* collection (OML's membership relationship) of tasks as performed by producers, collectively called task performances. Thus a workflow involves both tasks selected from one or more activities and producers. These can be identified differently for different purposes, such as to show the complete flow of tasks involved with the creation, iteration, evaluation, and delivery of a work product. In contrast, Activities are seen as heterogeneous collections of Tasks in which there is no notion of sequencing.

Finally, the larger scale picture is shown in **Figure 10** in which we can see how the OPEN process framework is tailored to suit individual projects, using elements from the instantiation and tailoring guidelines, the existing suite of process framework components (e.g. the Techniques described in the OPEN Toolbox: ref. 10), consisting of predefined work products, producers and work units.

Summary

This new, improved version of the OPEN metamodel significantly improves and extends its capabilities. There is a new emphasis on work products and their (typically human) producers. There is a more object-oriented feel to the metamodel with significant abstractions of the work products and the teams producing those work products rather than a sole focus on reified operations (such as activities and tasks) — the latter having

been reified in order to provide flexibility in tailoring and instantiation (of a specific process from the metamodel/framework). This re-orientation also permits the metamodel itself to be coded (we already have a test version written in Java) to produce an executable framework so that the process engineer can use this software to simulate possible process options.

In addition, the new metamodel offers facilities directed towards the current OMG's RFI on process. The metamodel describing the OPEN framework, as outlined here, would appear to be an ideal candidate to fulfil the requirements of this new standardization process.

Acknowledgements

This is Contribution no 99/15 of the Centre for Object Technology Applications and Research (COTAR).

References

1. Henderson-Sellers, B., 1999, A methodological metamodel of process, 1999, *JOOP/ROAD*, **11(9)**, 56–58, 63

2. Henderson-Sellers, B., Firesmith, D.G., Graham, I. and Simons, A.J.H., 1999, Instantiating the process metamodel, *JOOP/ROAD,* **12(3)**, 51–57

3. Henderson-Sellers, B. and Mellor, S., 1999, Tailoring process-focussed OO methods, *JOOP/ROAD*, **12(4)**, 40–44, 59

4. Graham, I., Henderson-Sellers, B. and Younessi, H., 1997, *The OPEN Process Specification*, Addison-Wesley, UK, 314pp

5. Graham, I. and Henderson-Sellers, B., 1999/2000, paper in preparation

6. Firesmith, D.G., 1999, An object-oriented software development process framework, *TOOLS 20* (eds. D.G. Firesmith and B. Meyer), IEEE Computer Society Press, Los Alamitos, USA (in press)

7. Nesi, P. (ed.), 1999, *Computer Science Dictionary (Software Engineering Terms)*, CRC Press, Boca Raton, FL, USA (in press)

8. Gustafsson, B., 1999, Rational Unified Process — process architecture and UML, presentation at UML World, New York, March 1999

9. Firesmith, D.G., 1999, *Object-Oriented Development Process Framework Specification*, AWL (in preparation)

10. Henderson-Sellers, B., Simons, A.J.H. and Younessi, H., 1998, *The OPEN Toolbox of Techniques*, Addison-Wesley, UK, 426pp + CD

Table 1

Terminological Definitions

- **Activity** —a cohesive collection of task performances that summarizes how the producers collaborate to create work products
- **Application** — a fully functional work product to be released to a user organization
- **Assertion** — a Boolean condition used (here) to control work units in a contract-driven development cycle
  - — **Invariant**: an assertion associated with a producer that must be true both before and after the successful execution of all associated work units
  - — **Postcondition**: an assertion associated with a work unit that must be true after the successful execution of that work unit
  - — **Precondition**: an assertion associated with a work unit that must be true before that particular work unit can successfully execute
- **Build** — a major increment of the entire application that is scheduled and formalized by a management team. A build tends to be large and relatively infrequent (e.g., monthly, quarterly, biannual)
- **Deliverable** — an attribute of Work Product denoting that the work product versions will be released to the customer or user
- **Development Cycle** — a cohesive collection of two or more development activities performed on a project
- **Development Process/Development Method** — the combination of work products, their producers and the languages in which they are expressed, and the activities that summarize the performance of tasks by collaborating producers using techniques
- **Development Process Framework** — a class library of predefined process components (i.e., the metamodel) and the instantiation guidelines that describe how process engineers and methodologists can use and extend them
- **Diagram** — a graphical work product representing a part of a model, or a view of that model, consisting of a directed graph of nodes connected by arcs representing relationships between the nodes
- **Document** — a kind of work product consisting of official written (and possibly graphical) information about one or more related work products
- **Drop** — a minor increment that is scheduled and formalized by a technical team. A drop tends to be small (only part of an application) and frequent (e.g., daily, weekly, monthly)
  - — **Code Drop** — a drop consisting of versions of software work products
  - — **Document Drop** — a drop consisting of versions of document work products
- **Endeavour** — a high level venture (programme or project) that develops one or more related software applications
- **Increment** — a collection of work product versions that is formalized at a given point in time (i.e., milestone of development)

- **Metric** — a work product measuring some aspect of something (e.g., progress, quality of work product)
- **Model** — a work product representing some part of the application or application domain. A model is an abstraction, capturing essential aspects while ignoring diversionary details
- **Person** — a human who plays a rôle in the development process
- **Producer** — someone or something actively involved in the production (e.g., creation, modification, evaluation) of one or more versions of work products
  - **Direct producer**: a producer that directly produces work products
  - **Indirect producer**: a producer that indirectly produces work products
- **Programme** — a collection of related projects managed as a unit
- **Project** — the enactment of a development process (i.e., execution over time) to produce one or more versions of a "major" work product (typically an application)
- **Release** — a build that is transferred to another team or organization (typically an external customer or user organization)
- **Requirement** — a work product that specifies a mandatory need of the customer or user with respect to the application
- **Responsibility Card** — a work product that documents the responsibilities of something (e.g., class, package)
- **Rôle** — a cohesive part that is played by a person during the production of one or more versions of the work products
- **Schedule** — a work product that captures the estimated and actual dates and time utilization associated with a task, activity, work product, rôle or team
- **Software** — Electronic portion of programs, procedures, associated data, operations, rules and associated documentation of an information processing system (ref. 7)
- **State Machine** — model of how an entity changes state. Often represented by a State Transition Diagram with nodes representing states and directed arcs representing transitions between the states
- **Task** — a work unit consisting of the reified operation performed by a direct producer
- **Task Performance** — the execution of a technique that implements some task performed by a producer
- **Team** — a cohesive group of rôles and/or other teams
- **Technique** — the reified implementation (i.e., how) of a task using the Strategy Pattern. In order to provide flexibility during tailoring and instantiation of the framework, producers may delegate the performance of their own tasks to techniques
- **Tool** — an application that is used to produce versions of work products
  - **Documentation tool**: a natural language tool for creating and maintaining documents
  - **Lower CASE tool**: a CASE tool for creating, compiling, debugging, testing and managing software
  - **Upper CASE tool**: a CASE tool for creating, evaluating and managing models

- **Work Flow** — a sequence of contiguous task performances whereby producers collaborate to produce work products
- **WorkUnit** — a functionally-cohesive operation that is performed during development and reified as an object in order to provide flexibility and support tailoring
- **Work Product** — something of value produced or used during development (e.g., document, diagram, application, class). During the evolution of a work product, work product versions are created. Therefore, each work product has an associated, time-ordered set of work product versions
- **Work Product Version** — a uniquely identified and dated occurrence of a work product. A producer creates a work product version while performing a task during incremental or iterative development. A version of a work product may be deliverable or only produced for internal use

Figure Legends

Figure 1 The Software Engineering Process (SEP) metamodel (modified from ref. 1).

Figure 2 Activities, tasks and sequencing rules make up the OPEN process (in Version 1) and link to the Work Products (called Deliverables in Version 1) and Techniques of Figure 1 (after ref. 1) ©SIGS

Figure 3 Summary of Version 2 metamodel for the OPEN process framework (adapted from ref. 3)

Figure 4 Details of the development process metamodel (after ref. 9)

Figure 5 Details of the work products inheritance hierarchy (after ref. 9)

Figure 6 Details of the producer inheritance hierarchy (after ref. 9)

Figure 7 Software development team aggregation structure (after ref. 9)

Figure 8 Details of the development cycle and work units (after ref. 9)

Figure 9 Sequence diagram illustrating how metamodel components can collaborate to produce and baseline a version of the software requirements specification (an example work product)

Figure 10 The structure of a development process framework (after ref. 9)