By B. Henderson-Sellers, D.G. Firesmith, I. Graham, and A.J.H. Simons

Instantiating the Process Metamodel

In my last column,[1] I described proposals for a process metamodel. In this month's article, I am joined by colleagues from the OPEN Consortium in describing three instantiations of that process metamodel[1]—all of which turn out to be describable as variants of the full OPEN process framework[2]—together with a description of how Objectory/RUP also fits into this framework and how it too can be viewed as compatible with the OPEN methodological framework and described by its metamodel.

INSTANTIATING THE METAMODEL

The process metamodel[1] is an architecture together with a set of rules that govern how any particular instantiation of that particular metamodel *must* behave. The architecture we described previously consists of an amalgam of:

- people, their skills, and the organizational culture in which they work
- the hardware and software at their disposal
- the elements of a methodology, which is a set of rules and guidelines on how to manage the modeling process, together with instructions on how to effect those models (the techniques of object modeling)

Together these produce outputs or artifacts that have to be described and documented. This can be done in a combination of natural language and a specialized object-oriented modeling language like OML or UML.

The elements of the metamodel that need to be instantiated to create the modeling process are shown in Figure 1. Activities are needed to structure the process, i.e., the time dependency. These could be listed simply, and rules could be written down for their use. In the OPEN methodological framework, an instantiation of the process metamodel, we don't merely list the activities but create objects to represent each of them with sophisticated time sequencing governed by pre- and postconditions.

Figure 1 also describes the need to identify and list tasks and techniques. Again, how these are linked together is a feature of the particular instantiation of the method. In OPEN those are linked together using deontic matrices (as discussed in our May 1997 column[3]). Deliverables and workflows are then created from the metamodel using the metaconcepts of DELIVERABLES* and SEQUENCING RULES (see Fig. 1).

OPEN is an instantiation of the process metamodel, which I described in my last column.[1] While it is not the only instantiation, it does act as a methodological framework that needs to be tailored to create individual process methodologies appropriate for specific:

- organizational cultures (e.g., random, hierarchical)
- domain types (e.g., real time, MIS)
- actual domains (e.g., banking, control systems)

* Although not all of these are delivered to customers/users.

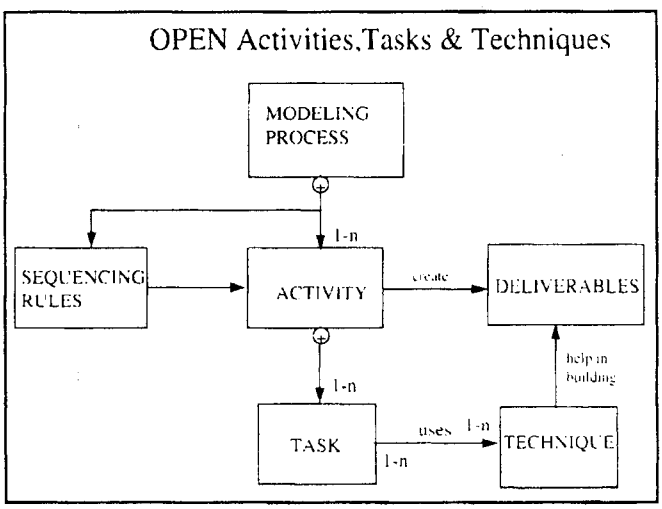

Figure 1. Activities (with possibly subactivities) have tasks that are realized by the use of one or more techniques (after ref. 1).

Brian Henderson-Sellers is a Professor of Information Systems in the School of Computing Sciences, University of Technology, Sydney (UTS), New South Wales, Australia. He can be contacted at brian@socs.uts.edu.au. D.G. Firesmith can be contacted at FiresmithD@aol.com. I. Graham can be contacted at 10170.3061@compuserve.com. A.J.H. Simons can be contacted at A.Simons@dcs.shef.ac.uk.

- safety levels required (e.g., life-threatening, research prototype)
- team sizes
- skills levels available in software developers
- available management skills
- required documentation levels and standards
- requirements to meet standards (e.g., ISO900x)
- deadlines (e.g., use of timeboxing and incremental delivery)
- available tools (CASE, compilers, and test suites)
- knowledge level of "object think"

This tailoring, particularly at the organizational and/or domain level, will create a process that an organization can adopt as its standard (assuming it continues to work in that domain with the existing culture, team sizes, etc.). Many industries that we work with do just that and create the Company X instantiation of OPEN. Also, increasingly, the developers and users of OPEN will create templates for chosen domains, team sizes, etc. In this article, we describe a small number of these: OPEN/SOMA focuses on MIS application development with a strong emphasis on requirements engineering; OPEN/Firesmith has been used particularly for embedded real-time software; and OPEN/Discovery provides a stronger emphasis on guided technical development and longer-term multiproject architectures.

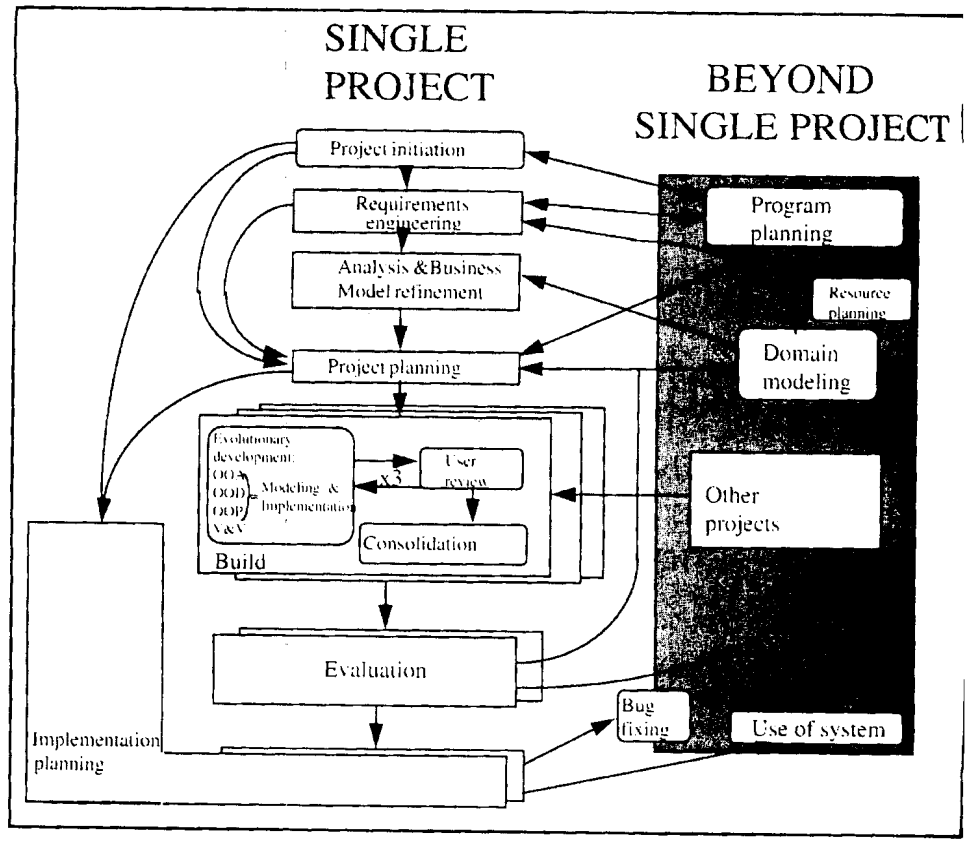

Figure 2. Contract-driven lifecycle process model—one example instantiation (revised from ref. 2).

OPEN'S CONTRACT-DRIVEN LIFECYCLE AND OPEN/SOMA

The contract-driven lifecycle is used in OPEN to describe the Activities and how they might be sequenced. The connections depend upon the domain—Figure 2 shows a set of connections that characterize those required for an MIS style of software development. This lifecycle instantiation is typical of that of OPEN/SOMA.

The model is, in essence, very simple. In summary, a development program may be decomposed into a network of projects that produce deliverable, working software products. Projects are composed of Activities (as detailed in *The OPEN Process Specification*[2]). The activities in OPEN permit a large-scale temporal structuring of the project management for OO product development, necessarily including schedules for Builds and Releases.

The progression order is neither prescriptive nor deterministic. Rather, the contract-driven lifecycle permits (1) ordering in an iterative, incremental, and parallel fashion and (2) the tailoring by a given organization to its own internal standards and culture. Different configurations, chosen to reflect the needs of different problems and different organizations, can be constructed from this flexible framework. We indicate in Figure 2 some of the likely routes between Activity objects for an MIS project; other variations are possible. The main governing constraint on the routes is whether or not you meet the preconditions of the Activity to which you wish to transition.

Each Activity has a set of goals and, in truly object-oriented fashion, has an interface consisting of methods, which are the selected Tasks; internal state and external services; and has pre- and postconditions to satisfy all contractual obligations. Each OPEN Activity is shown as a box, either with rounded corners (an unbounded Activity) or rectangular corners. (The latter represents activities tightly bound in time, i.e., the ideas of timeboxing are applied—see section 4.1 of *The OPEN Process Specification*.[2]) Because we are modeling these activities as objects, we can associate contracts with each Activity object (hence the lifecycle name of "contract driven"). These are expressed primarily by pre- and postconditions; in other words, constraints that have to be met before an Activity can be commenced, and final conditions that have to be met (and signed off) before the Activity is complete and another Activity can be initiated (triggered). Testing is an integral part of these exit conditions. Testing should deliver test results against use cases and/or task scripts (a higher form of use cases[4]) and a techni-

cal test plan of the normal kind—for example, tests that answer the questions: Does an enhancement leave previous things that worked working? Does the system work well under stress and high volume I/O?

More strictly, the contracts and pre- and postconditions are specified at the operation level, rather than the object level. In this lifecycle model, we consider the Tasks as the operations of the Activity objects such that it is the Tasks to which the pre- and postconditions are applied.

The contract-driven lifecycle allows the developer to link Activities in a specific time sequence (thereby creating a process). The Activities can in fact be linked in any order so long as all the pre- and postconditions are met. This does not, however, lead to anarchy. Rather, it supports flexibility so that different organizations can choose their own tailored lifecycle process model from the OPEN Activities. Secondly, within a single organization, individual projects may have different foci, which require different configurations of this highly flexible lifecycle process. Thus, an individually chosen lifecycle can be created for projects and/or organizations while remaining "OPEN compliant." The developers still speak the same language and can still talk to each other!

Each combination of Activities and interconnecting paths defines a software engineering process (SEP)—here the OPEN/SOMA SEP. Once chosen, the lifecycle process is fixed—although still, at least in an OO project, highly iterative, incremental, parallel, and flexible and with a high degree of seamlessness.

The OPEN/SOMA SEP closely parallels the general OPEN framework as it was applied initially in the MIS domain. More recently, however, OPEN/SOMA is better characterized as:

- being suitable for RAD and compliant with DSDM
- having a strong focus on business-process modeling
- enabling actions (use cases) to be derived from process models
- including strong requirements engineering and model-validation techniques
- being a useful precursor to component-design methods such as Catalysis[5]
- including techniques for AI development as well as conventional analysis

THE OPEN/FIRESMITH INSTANTIATION

While all OPEN instantiations adopt the broad construction principles discussed in more detail for OPEN/SOMA, the details can be very different, e.g., different Activities, different linkages between Activities, and so on.

In our second example instantiation of the process metamodel, we describe the OPEN/Firesmith standard instantiation of the OPEN methodological framework. Originally developed for real-time embedded applications, this method has been used in most application domains when properly tailored. Figure 3 shows the Activities of the OPEN/Firesmith method.[6] The Initial

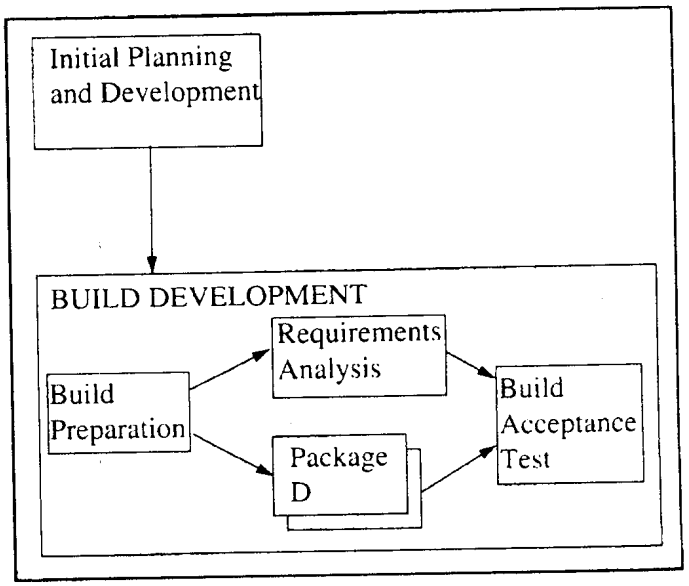

Figure 3. The Firesmith instantiation of OPEN: Planning and Build Development (after ref. 11).

Planning and Development Activity has Tasks that include Document the Background, Initial Training, Tailor the Method, Model the Context, Capture the Initial Requirements, Create the Initial Architecture, Document the Initial Products, Scope the Application, and Plan Build Development. While there is a rough ordering of these Tasks, there is also significant temporal overlap.

In the OPEN/Firesmith instantiation, the second major Activity is Build Development. In parallel with the Build Activity of Figure 2, this also has subactivities and is the focus of the technical part of software development. Following Build Preparation, Requirements Analysis and Package Development proceed in parallel. Build Acceptance Testing then completes the Activity. The Task Verify the Packages and Use Cases can be useful throughout, but note that "due to the functional nature of use case modeling, use cases should be used more for verification purposes than for requirements analysis purposes."[6]

Requirements Engineering and Package Development proceed in parallel. The Tasks associated with the former include

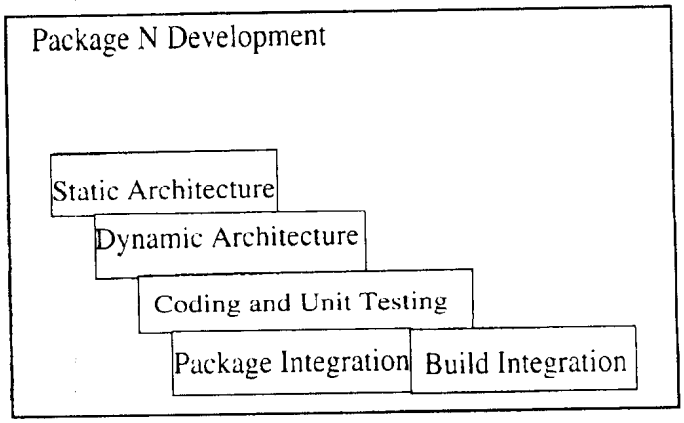

Figure 4. The Firesmith instantiation of OPEN: development of the *n*th package (after ref. 11).

Requirements Elicitation, together with Requirements Analysis and Specification. Package Development is highly iterative and for each iteration has several Tasks (see Fig. 4) that are performed in a highly iterative, incremental, and parallel fashion. Different packages can be developed concurrently by assigning them to different package-development teams. The appropriate Tasks include Identify the Packages, Model the Package Static Architecture, Model the Package Dynamic Architecture, Coding and Unit Testing, Package Integration, and Build Integration.

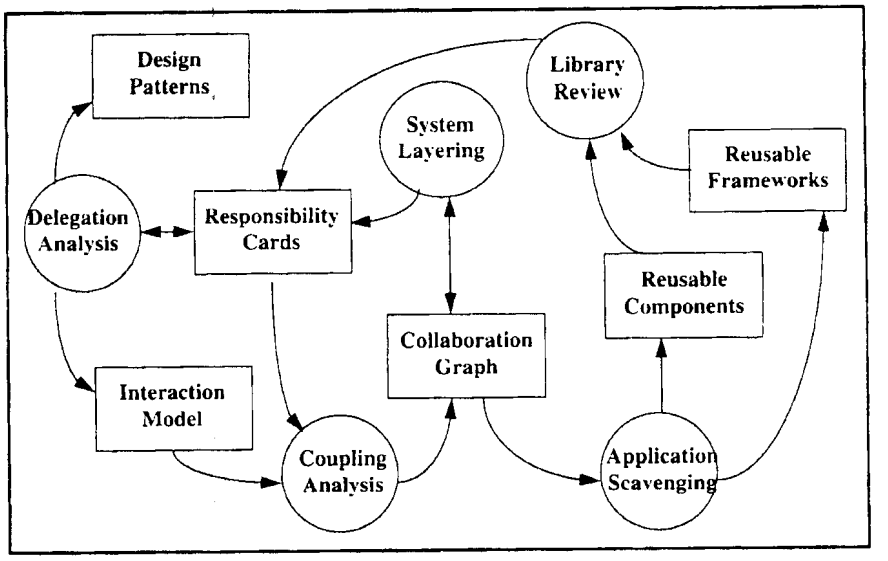

Figure 6. System modeling in OPEN/Discovery at the meta-level (after ref. 11). (See key in Fig. 5.)

DISCOVERY AS AN INSTANTIATION

In the third example, we move to a specific instantiation that provides a much more detailed emphasis on technical development: OPEN/Discovery.[7,8] Discovery has less over-arching process machinery and more detailed guidance on the application of individual techniques. Discovery "inlines" the OPEN three-tier process architecture (Activities, Tasks, and Techniques) so that developers can follow a roadmap of Techniques recommended for their fitness-of-purpose in helping to complete standard OPEN Tasks in analysis and design. Discovery's main four process stages are: Task Modeling (requirements engineering and project planning), Object Modeling (analysis and initial design), System Modeling (system-level design and framework reengineering), and Language Modeling (mapping final designs onto appropriate language constructs). This configuration represents a different partitioning of Tasks compared with the default OPEN Activity-to-Task mapping. For example, long term and cross-project framework design issues are considered in the same stage as the system-level design of the current application because of the way in which these concerns are closely related (see Fig. 6).

Figure 5 illustrates Discovery's Object Modeling stage, depicting a series of Techniques (in circles) and Deliverables (in rectangles) that are selected by Discovery to accomplish the OPEN Tasks: Identify CIRTs, Construct the Object Model, Design the Database, and Design the User Interface. Note that developers do not need to be aware of the correspondences between OPEN Tasks and Techniques to use Discovery effectively—this mapping is decided by the designers of the Discovery method. In this way, Discovery is a particular instantiation of the OPEN process model. Figure 5 shows the major dependencies between Techniques and the Deliverables they generate as a directed graph. Dashed outlines indicate Techniques that are optionally applied in some projects for which they are appropriate. For example, Storage Analysis is a Technique that may be applied if a large back-end database is required, but this is not the case in every object-oriented project. Storage Analysis can proceed once the domain Vocabulary is stable and the Task Scripts are complete. This is because the nouns chosen for data entities should correspond to stable domain concepts; and the way data files are partitioned depends crucially on how data is to be used by the client. The output of Storage Analysis is a Data Model, structured relational tables, or persistent object files, minimally connected through foreign keys or pointers, respectively. The dependencies illustrated correspond to the pre- and postconditions in the OPEN contracting model, determining which Techniques may be applied when. This explicit dependency does not preclude iterative development (a fountain-like approach in which the arrival of new Scripts and Vocabulary might trigger a further round of Storage Analysis and optimization of the Data Model), but is there

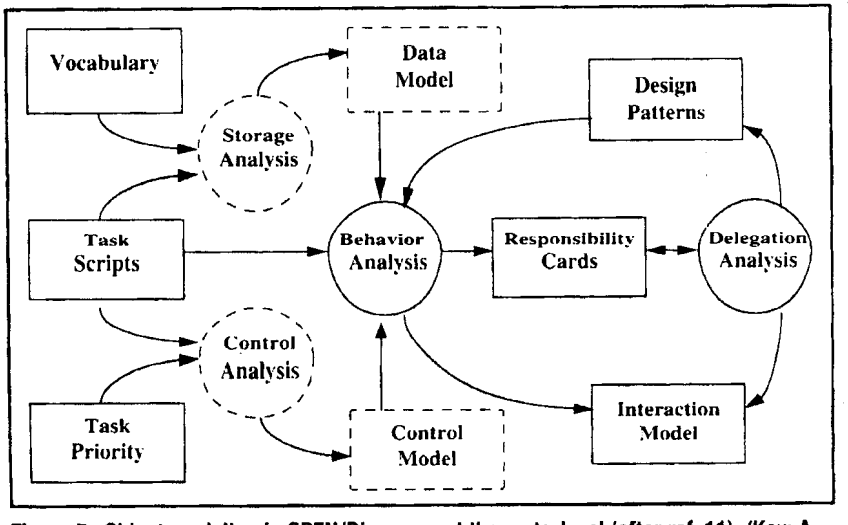

Figure 5. Object modeling in OPEN/Discovery at the meta-level (after ref. 11). (Key: A rectangle represents a model to be delivered, a circle is the process of applying a technique, an arrow indicates the direction of information flow, and a dotted outline indicates optionality.)

to help developers understand how the stages of system development depend on each other. The diagram illustrates how certain Techniques could be applied concurrently. For example, Control Analysis could proceed in parallel with Storage Analysis. There are also major feedback loops in the process. For example, Delegation Analysis is a technique for refining candidate object concepts recorded in the Responsibility Cards. It sometimes splits an object into equal peers, or breaks an object into a master and a subcontractor. Not only does this affect the base set of Responsibility Cards, but also reveals new instances of design patterns such as "Chain of Responsibility," which are then useful for establishing the right mindset for Behavior Analysis in this and subsequent projects. Delegation Analysis also restructures a subset of the Interaction Models to reflect the local changes in the patterns of messaging.

The OPEN process model describes many possible mappings from Tasks to Techniques. The Task is the goal, the "what"; the "how" is described by one or more Techniques. In general, a developer may have to choose between several possible techniques, some of which may be useful all the time or just possibly useful, while other techniques should be avoided altogether. A particular instantiation of OPEN, such as Discovery, is free to recommend some techniques over others for particular purposes. One of the main principles of the Discovery method is that every technique has a single focus, stimulating a particular mindset in the developer: the "cognitive bias." A technique should therefore not be pressed into service to accomplish something for which it was not intended. For example, the entity-relationship modeling that is undertaken during Storage Analysis is only used for database table design and not especially for object identification, which is carried out using a responsibility-driven approach (see Fig. 5). This is because minimizing data dependency and minimizing client-server dependency are two different kinds of processes; objects should therefore not be constructed simply by adding methods to the entities found in ER tables. In general, Discovery takes a strong stance against "universalist" approaches that try to propagate the same models through the analysis, design, and implementation stages. Systems that are constructed this way exhibit poor coupling characteristics.[9,10] Instead, Discovery mandates a series of system-level transformations in the System Modeling stage (see Fig. 6), in which client-server collaborations are analyzed at the class level and reworked with the introduction of new concepts corresponding to the Mediator, Command, and Template Method Patterns. The method generates these concepts automatically, using simple constraints to identify appropriate subsystems and reduce interclass coupling. The mutual influence of the current emerging design and existing frameworks is properly addressed (see Fig. 6). These techniques used by Discovery and many more from the other methodologies contributing to OPEN are described in the OPEN Toolbox of Techniques.[11]

A technique should therefore not be pressed into service to accomplish something for which it was not intended.

OBJECTORY/RUP AS PART OF THE OPEN FRAMEWORK

Another process proposed for use in object-oriented software developments is Objectory, which went through a number of versions up to Version 4.1 in 1997. At that stage, the process elements, such as they were, from the Booch methodology and from OMT were merged into it, resulting in the Rational Unified Process (RUP)—although it is not clear whether this may simply be the name of the accompanying product/tool*—which still bears a strong similarity to the original Objectory. It is declared to be use-case driven, architecture centric, and iterative/incremental. It is also said[12] to have the three poles of People, Process, and Tools in agreement with the suggestions in Figure 1.2 in *The OPEN Toolbox of Techniques*.[11]

* The statement that the "Rational Way" is highly automated[12] supports this notion.

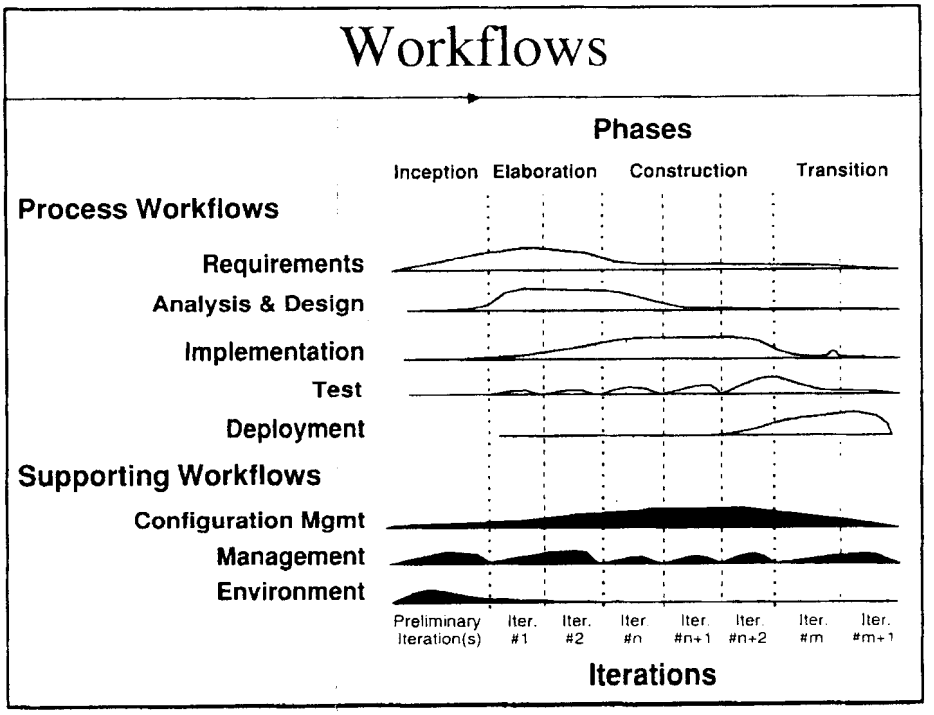

Figure 7. Workflows within the RUP process (after ref. 12, ©Rational, 1998).

Jacobson[13] describes the Unified Process as having four phases: Inception, Elaboration, Construction, and Transition. These would seem to correspond to the concept of Activity in the metamodel of Figure 1 and are described by Kruchten[12] as giving the management perspective. The RUP Inception phase appears to map to the Project Initiation activity of Figure 2; the RUP elaboration phase subsumes requirements engineering, model refinement, and project planning. Construction and Transition are equivalent to the Build Activity and Implementation Planning, respectively. There is no equivalent to the Evaluation Activity nor to any activities in the Program part of Figure 2. Thus, RUP is a different instance of the metamodel of my last column[1] in which there is an easily identified correspondence with the activities in the OPEN-compliant methodologies discussed earlier.

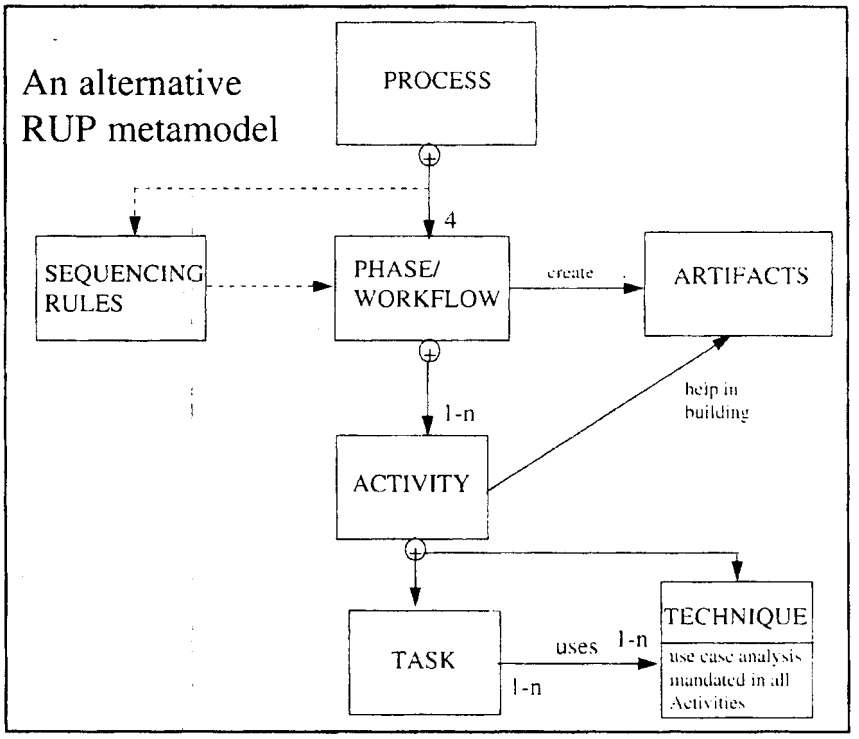

Figure 9. An alternative view of the RUP metamodel (derived from description in ref. 17).

A number of "workflows" such as requirements, analysis, and design are superimposed across these four phases (see Fig. 7). Because these are cross phase/activity boundaries, they would initially appear to map to Tasks. However, their granularity is more like that of subactivities, e.g., in Figure 2, analysis and design are part of the Modeling and Implementation subactivity. The idea of a fuzzy linkage is there but expressed in a slightly different way in terms of temporal effort curves (a technique first used to describe OO lifecycles in "The Object-Oriented Systems Life Cycle"[14]). This is the technical perspective.[12] Together the management and technical perspectives show the influence of Booch's[15] macro and micro lifecycles.

These workflows are bound together by use cases. Thus, we can model RUP in exactly the same way as the other methods described here, as an instantiation of Figure 1,[1] but now with an additional constraint that all Tasks must link to a specific Technique: use case analysis. In other words, this is a special case of OPEN's 2D matrix in which some of the links (those involving use case analysis) are predetermined, fixed, and *not* able to be tailored by the user.

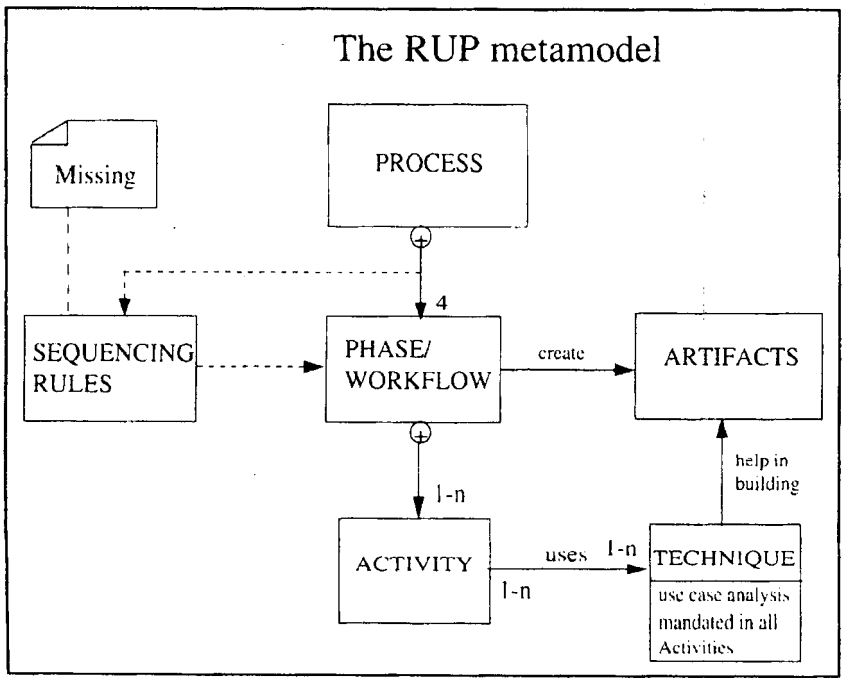

Figure 8. The RUP metamodel (contrast with Fig. 1). The dotted lines suggest connections that would be anticipated to exist but are in fact missing.

Jacobson[13] has also stated that an "Activity" is "a unit of work a worker may be asked to perform." This is exactly the definition of Task in my last column[1] and in *Succeeding with Objects: Decision Frameworks for Project Management*.[16] So an RUP Activity maps precisely to the metamodel (and to OPEN's) Task (see Fig. 8), although according to *The Unified Modeling Language User Guide*,[17] an RUP Activity is an aggregate of Tasks and Techniques (see Fig. 9). Finally, the Deliverable of Figure 1 is called Artifact in RUP.

What seems to be missing from RUP, as described in these sources, are the sequencing rules and explicit Techniques (see Fig. 1). Nevertheless, the mapping to Figure 1 as shown in Figure 8 (or Figure 9) and a comparison at the "instance level" of Figure 10 with Figure 2 are both in-

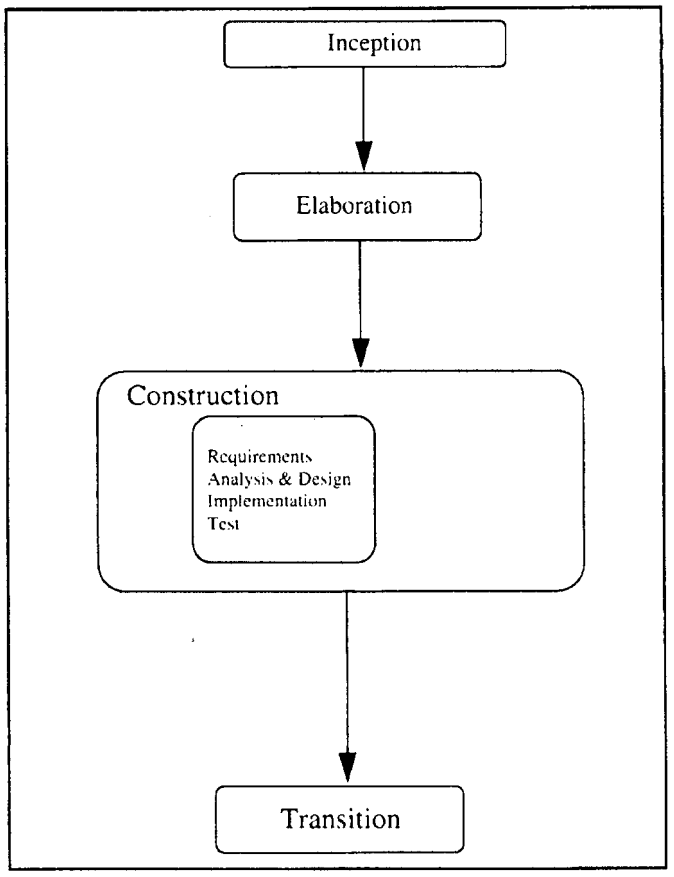

Figure 10. The RUP instantiation (contrast with Fig. 2).

dicative that RUP can be generated by the process metamodel under discussion and described in detail in my last column.[1]

The way in which the architecture-centric nature of RUP is built in can be done in a way that's identical to OPEN (through the use of appropriate tasks and techniques). However, what is not as clear is the way that the iterative nature of OO development is used. In RUP (as seen in Figure 8), iterations are confined within a specific phase and describe increments. This is more akin to the more sophisticated Waterfall models than a truly configurable and fully iterative (across the full lifecycle) process.

While the process as derived in Figure 10 is essentially a Waterfall one with embedded iterations (increments), Kruchten[12] also describes a number of Entry and Exit criteria for each phase. These are identical in spirit to the pre- and postconditions of the contract-driven lifecycle,[2,18] although the notion of contracting per se is not mentioned by Kruchten.[12]

CONCLUSION

Instantiating a process metamodel can create different versions of that metamodel tailored to specific organizations, projects sizes, capability, etc. Based on the OPEN process metamodel,[1] we have shown how four existing process models (SOMA, Firesmith, Discovery, and RUP) are easily instantiated from this metamodel. The metamodel thus provides an excellent starting point for creating tailored versions of a process, either pretailored by methodologists (as illustrated here) or by individuals within your own organization to create a company-specific methodological process that mirrors the existing and developing organizational culture. This prevents the current approach of either force-fitting the methodology to the culture or, more commonly, pretending to adapt the culture to the methodology. In the latter case, this usually results in methodological shelfware. Creation and adoption of the tailorable OPEN framework obviates this danger admirably. ■

Acknowledgments

We wish to thank Colin Atkinson for comments on the OPEN to RUP mapping. This is Contribution no 99/1 of the Centre for Object Technology Applications and Research (COTAR).

References

1. Henderson-Sellers, B. "A Methodological Metamodel of Process." *JOOP* 11(9):45–55, Feb. 1999.
2. Graham, I., B. Henderson-Sellers, and H. Younessi. *The OPEN Process Specification*. Addison-Wesley, London, UK, 1997.
3. Henderson-Sellers, B., I. M. Graham, and D. Firesmith. "Methods Unification: The OPEN Methodology," *JOOP*, 10(2):41–43,55, May 1997.
4. Graham, I. "Some Problems With Use Cases ... and How to Avoid Them," in *OOIS '96 Proceedings*, D. Patel, Y. Sun, and S. Patel, Eds., Springer-Verlag London Limited, London, pp. 18–27, 1997.
5. D'Souza, D. F. and A. C. Wills. *Objects, Components, and Frameworks with UML:The Catalysis Approach*, Addison-Wesley, Reading, MA, 1999
6. Firesmith, D. G. et al. *Object-Oriented Development Using OPEN: A Complete Java™ Application*. Addison-Wesley, London, 1998.
7. Simons, A. J. H. "Object Discovery—A Systematic Approach to Developing Object-Oriented Systems." Workshop 8. *BCS Conference on Object Technology*. Apr. 1998.
8. Simons, A. J. H. "Object Discovery—A Process for Developing Medium-Sized Object-Oriented Applications." Tutorial presented at *ECOOP*, July 1998.
9. Simons, A. J. H., M. Snoeck, and K. S. Y. Hung. "Design Patterns as Litmus Paper to Test the Strength of Object-Oriented Methods, *OOIS '98 Proceedings*. C. Rolland and G. Grosz, Eds., Springer-Verlag, London, pp. 129–147, 1998.
10. Simons, A. J. H. and M. Snoeck. "Rigorous Object-Oriented System Design." *Proc. 2nd Conf. Rigorous Object-Oriented Methods*, Bradford, 2.1–2.24, 1998.
11. Henderson-Sellers, B., A.J.H. Simons, and H. Younessi. *The OPEN Toolbox of Techniques*, Addison-Wesley, London, 1998.
12. Kruchten, P. A Rational Development Process, (unpubl. 1999); available from www.rational.com/products/rup/whitepapers/.
13. Jacobson, I. "The Unified Process," keynote presentation at *EDOC'98*.
14. Henderson-Sellers, B. and J. M. Edwards. "The Object-Oriented Systems Life Cycle." *Comms. ACM*, 33(9):142–159, 1990.
15. Booch, G. *Object-Oriented Analysis and Design with Applications*, 2nd ed., Benjamin/Cummings, Redwood City, CA, 1994.
16. Goldberg, A. and K. S. Rubin. *Succeeding with Objects: Decision Frameworks for Project Management*. Addison-Wesley, Reading, MA, p. 144, 1995.
17. Booch, G., J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, p. 454, 1999.
18. Graham, I. M. "A Non-Procedural Process Model for Object-Oriented Software Development." *ROAD*, 1(5):10–11, 1995.