Patterns of Workflow Management Facility

Dragoş-Anton Manolescu and Ralph E. Johnson
{manolesc,johnson}@cs.uiuc.edu

November 1, 1998

Abstract

The OMG has issued a Request for Proposal for a workflow management facility for the Object Management Architecture Reference Model. The two responses are complex object-oriented designs. Unfortunately, they both ignore recent work in object-oriented design patterns. Using these patterns will correct some of the deficiencies in the proposals.

1 Introduction

One of the areas subject to the Object Management Group's (OMG) standardization efforts is workflow. In 1997, the OMG issued a Request for Proposal (RFP) for a workflow management facility [OMG97]. Once adopted, the facility will be part of Object Management Architecture Reference Model.

Two submissions have been received in response to this RFP. One submission is provided by Nortel and supported by the University of Newcastle upon Tyne [OMG98b]. The other is a joint submission (jFlow) from about 20 different companies [OMG98a]. Each proposal describes an object-oriented solution for the workflow facility. They focus on IDL specifications, UML diagrams and specify the mandatory, as well as some of the optional interfaces.

Software solves an increasing number of problems. Consequently, its complexity has increased as well. Understanding and maintaining software is both difficult and expensive. The object-oriented community has been exploring ways to keep complexity and costs under control. Things like "reuse software engineering" are becoming a business [JGJ97]. However, these techniques are hopeless without a good understanding of the problem, its solution and the available alternatives.

A workflow facility brings together principles, methodologies and technologies from various areas of computer science and management science [Moh97]. Unfortunately, the authors of the RFPs have not chosen the appropriate methods to present their designs. Consequently, the designs are hard to understand.

This paper is a critique of the design methodology adopted in the proposals. We employ a different technique which has been successful for building reusable object-oriented systems. Our instruments for documenting designs are software patterns [GHJV95, BMR$^+$96]. Patterns help describe complex systems and provide good, proven solutions. They also constitute a common vocabulary for abstractions. Documenting software with patterns is compact and unambiguous. Developers are aware of the benefits and liabilities of each solution. It is easier to identify design flaws early on.

This paper is organized as follows. Section 2 goes through the design of the facility. We first present the solutions adopted in the Nortel and jFlow RFPs. Next we describe what patterns are applicable, how they improve the solution and shorten the presentation. In contrast with the approach adopted by the authors of the original documents, we emphasize design. Section 3 asks a few questions about the current workflow RFPs. We believe that these issues need to be addressed in the forthcoming versions of these documents. We also provide several suggestions for improvement. Section 4 contains a summary of the patterns presented in this paper. Finally, we draw conclusions in Section 5.

2 Designing a Workflow Management Facility

A workflow system consists of two tiers. The top tier (referred to as "workflow process" in [GT98]) consists of rules that automate job coordination, control and communication, as well as resource allocation. Likewise, the lower tier consists of executing jobs[1] and resources. This tier is domain dependent, e.g., insurance, banking, administration, manufacturing, etc.

For example, a typical job in an insurance system is selling life insurance policies. One rule for this kind of policies may require human intervention for any life policy over \$1,000,000. In this case, operating on policies belongs to the domain dependent work tier. Coordinating these operations and passing results between jobs belongs to the flow tier.

A programming language is a potential tool for job and resource management—the flow part. Some load share systems use specialized languages for this purpose (e.g., JCL). Slow, long-running jobs and frequently

[1]Although the workflow reference model [Jol95] employs the term "process," we deliberately use "job." This makes it easier to distinguish from the Nortel and jFlow solutions, which use "task" and "process."

changing rules are typical characteristics of workflow management systems. Resources consist of IT applications or humans. At runtime, humans may even take over job ordering and coordination. Consequently, managing this kind of jobs is not suitable for a programming language.

2.1 Changing the rules

Business processes are dynamic. The types of jobs managed by the workflow management facility evolve along with the business system. Sometimes, new job types are introduced. For example, a telecommunication billing system may introduce a "rate the ASDL usage for the current billing cycle" job. Other times, existing job types are modified or even retired. For instance, an on-line bookseller may decide to provide a list of recommendations for each customer. Several additional jobs are created to support this feature: "set up customer profile," "collect recommendations" and "display recommendations." Since jobs usually take a long time to complete, this sort of changes may occur while they are running.

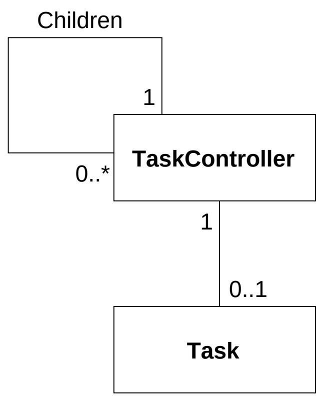
Modification of workflow rules may have local or global (temporal) scope. Local changes affect the jobs that start to execute after the changes are in effect. For example, modifications to the enrollment procedure for an insurance policy affect all policies issued after the new rules become effective. In addition, global changes also affect the jobs that are currently running. For the insurance domain, some changes in legislation are likely to have this effect. A successful workflow facility accommodates evolution transparently.

Job classification

The solution adopted by both RFPs is to use a process template for each type of running process. For example, the template for "install telephone service for subscriber" contains the activities and the resources for this process. It also contains the rules governing process execution.

The Nortel proposal employs a dedicated *task controller* for every task within the workflow application. Each controller receives and sends notifications, maintains the local structure of the workflow and provides information about the status of the task. Controllers can also be reconfigured to a different type of task (e.g., simple, compound, genesis or adapter). Operations for modifying and examining the task definition associated with a genesis task are also provided. Figure 1 shows the UML class diagram (top) and an instance diagram (bottom) corresponding to this solution.

The jFlow proposal suggests a slightly different approach. The process definition is encapsulated in a *manager* object. This represents a "template for a specific workflow process." The manager creates and ini-



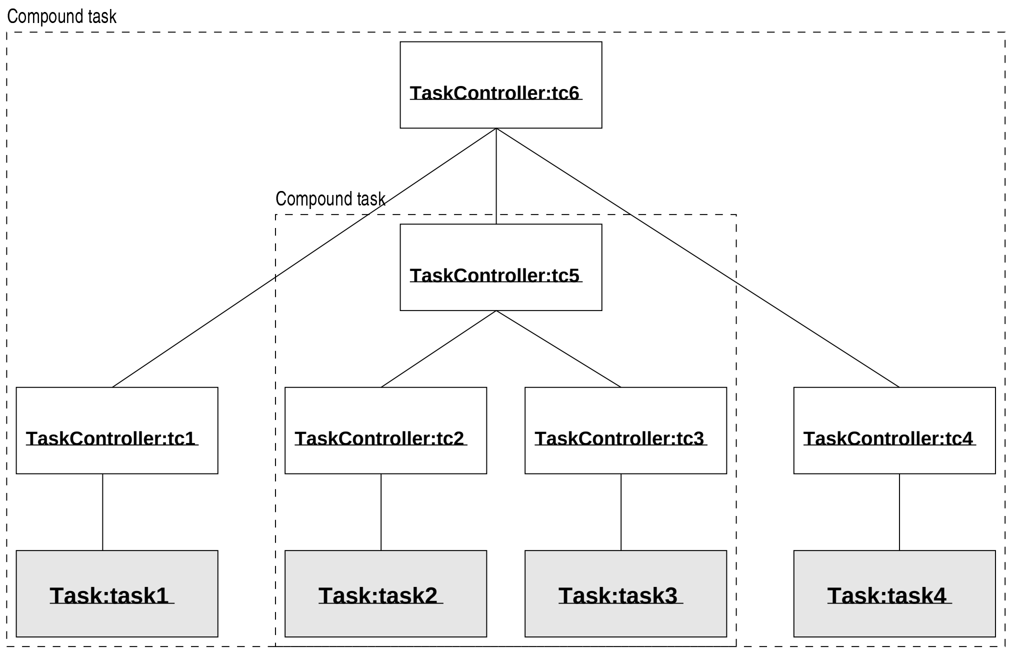


Figure 1. Class diagram (top) and an instance diagram (bottom) for the Nortel task model.

tializes instances of workflow processes. It also provides access to meta information and the result of the process. Figure 2 shows the UML class diagram corresponding to this solution.

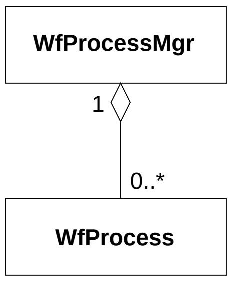


Figure 2. jFlow process model.

A `Task` and a `WfProcess` are similar. Let's call both of them *jobs*. A *job type* object is associated with each kind of running job. The job type contains some general characteristics and workflow rules. Therefore, the relationship between the job type and its corresponding running jobs is similar to the one between a class and its instances. Each running job has a particular configuration and a set of rules. Local changes correspond to changes at the instance level. Changing a running job instance does not affect the others. This kind of changes can be seen as "dynamic subclassing." Likewise, global changes correspond to changes at the class level. There, modifying the job type affects all the operations that instances delegate back to the type.

The Type Object pattern [JW97] solves the problem of decoupling instances from their classes. TypeObject has separate classes for instances and their types. In this case, the class of the instance would be `Task` or `WfProcess`, and the class of the type would be `TaskController` or `WfProcessMgr`. It appears that `WfProcessMgr` really is a type, since one instance of `WfProcessMgr` is associated with many instances of `WfProcess`. However, each `Task` has its own `TaskController`, so the Nortel task model does not use this pattern.

In the context of the workflow management facility, Type Object represents the core of the job model. Business rules are located on the type side, while the runtime information of each running job is located on the instance side. Type Object solves several workflow problems:

- The facility can create new job types at runtime. These are actually instances of the `JobType` class. Like in many object-oriented frameworks, a great deal of customization is possible only by creating new instances.

- `JobType` objects employ a Factory Method [GHJV95] to create `Job` instances. The factory can customize each instance it creates. Therefore, it is possible to have a wide variety of `Job` objects and only a small number of `JobType`s.

- Humans and IT applications that interact with the workflow system are not aware of the separation between a `Job` and its corresponding `JobType`. Clients interact only with `Job` objects and these forward some requests to the type side. The `JobType` implements common functionality (e.g., persistence, etc.), while each `Job` implements instance-specific details.

- The classification relation between instances (`Job`) and types (`JobType`) is under user control. Consequently, it is possible to change a job's type at runtime. This allows users to alter the behavior of executing jobs.

Recent studies discuss how business process modeling tools (BPMTs) and workflow management systems (WfMS) can interoperate to provide complete support for the entire business process lifecycle [GT98]. We believe that this job model facilitates the integration with BPMTs. In one direction, process definitions are translated into job types—interface 1 in the reference model. In the other direction, the type side provides runtime information about its corresponding job instances—interface 5 in the reference model. This is illustrated in Figure 3.

The jFlow process model from Figure 2 is similar to our job model. However, since its developers do not refer to the Type Object pattern, we are not sure whether they are aware of all the consequences of this design.

The parts and the whole

The job model presented so far does not provide details about job structure. A job consists of a number of discrete job steps. Let's examine first how the RFPs represent this structure.

The Nortel submission supports four distinct forms of tasks: simple, compound, genesis and adapter. Users combine these forms to specify and execute their business processes. Compound tasks represent the composition of a task out of other tasks. A compound task is actually a controller that controls other controllers, which in turn control task objects. The task controllers `tc5` and `tc6` from Figure 1 correspond to compound tasks. Figure 4 shows the UML class diagram for the Nortel task types and illustrates the recursive structure that the compound task introduces.

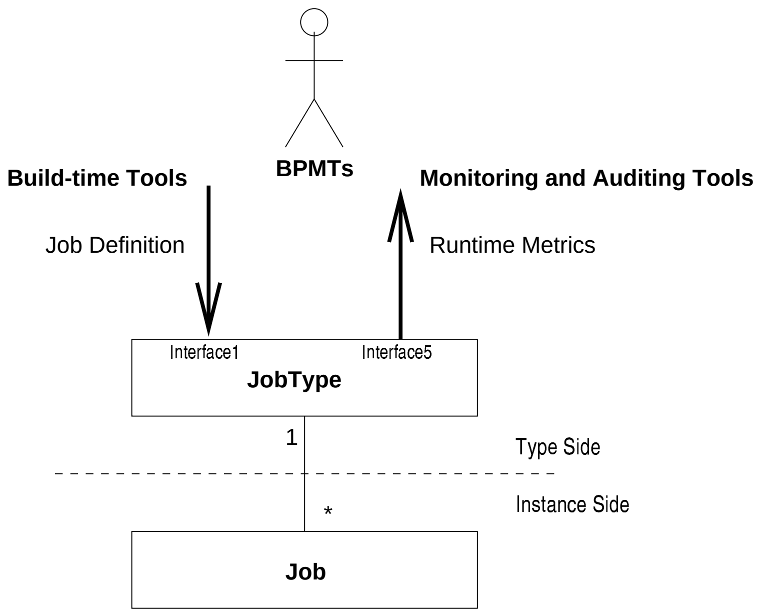


Figure 3. The interaction between BPMTs and the job model. BPMTs are depicted as an actor. The two interfaces are defined by the workflow reference model.

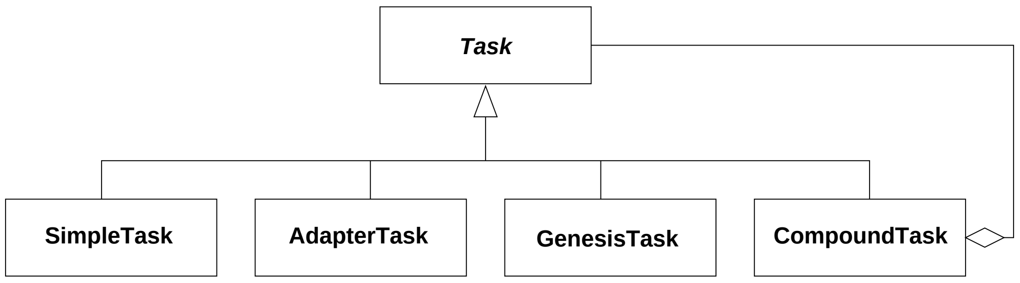


Figure 4. Task types in the Nortel RFP.

The jFlow submission represents processes as a series of steps. Each step corresponds to an activity. A recursive structure is possible as well, since an activity may be implemented by another process. This yields either nested or chained sub-processes. Therefore, the relationship between process and activity is containment. Figure 5 depicts the UML class diagram.

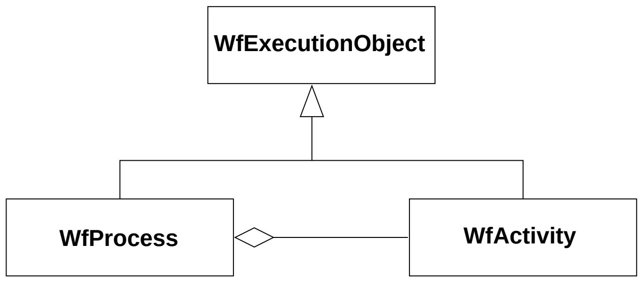


Figure 5. Process and Activity in the jFlow RFP.

Both proposals recognize the importance of assembling several parts into a whole. To see what design choices are available and make sure that we don't miss any significant details, we turn back to software patterns.

The Composite pattern [GHJV95] provides a design solution for this kind of problems. It allows clients to navigate a hierarchy and interact with individual objects and compositions in an uniform manner. The key idea of composite is to put the common interface in a base class. Figure 6 illustrates the corresponding UML class diagram.

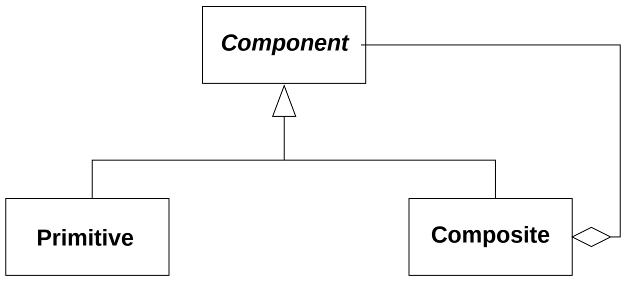


Figure 6. The Composite **design pattern. The** `Component` **class is usually abstract.**

In the context of the workflow management facility, job steps are the primitive components. Jobs are composites and consist of a sequence of steps. Recursive composition allows the facility to keep complexity under control—complex jobs are built from simple components. Moreover, this design choice provides additional flexibility whenever workflow rules change. Here are a few characteristics that make Composite a good choice for WfMS:

- Components are interchangeable. One type of component can substitute the other, at build-time as well as at runtime.
- Since jobs and job steps have the same interface, clients treat them uniformly. This helps simplify the clients.
- New components conform to the existing interface. No changes are necessary on the client side whenever users add new jobs or job steps.

Our design choice is similar with the structure from the Nortel proposal—Figure 4. This time we are in agreement with the RFP that does not use Type Object. The combination of the two patterns would yield a powerful and flexible design.

2.2 Overriding the rules

At runtime, the workflow enactment service manages jobs and invokes the appropriate human and IT application resources. This proceeds according to the job definitions provided by the build-time functions. However, sometimes the workflow rules may need to be overridden. Moreover, there are circumstances when a job's definition is completed while the job is executing.

This situation corresponds to ad-hoc workflows. These are useful when dealing with exceptions or unique situations [AAAM97]. In such circumstances, humans take over task ordering and coordination [GHS95]. Consequently, a WfMS that supports ad-hoc workflows has different requirements. The focus is on functionality for facilitating human coordination, collaboration and co-decision.

These requirements illustrate the unsuitability of programming languages to manage ad-hoc workflow. Only a few languages support this mode of operation. For example, in Smalltalk it is possible to pop up a debugger window, modify the code, change variables and then resume execution. However, most workflow users are not programmers and do not want to interact with the WfMS at this level.

Currently, several commercial products allow executing workflow instances to be dynamically modified [Moh97]. However, support for ad-hoc workflows is an optional requirement in OMG's RFP. Therefore, the Nortel and jFlow proposals provide only a few details about this issue.

One of the advantages of a pattern-based facility is that it is easy to asses its suitability for some given requirements. A few characteristics that provide support for ad-hoc workflows follow:

- Type Object allows a running job to dynamically change its job type. Finer granularity changes are possible by modifying the delegation from the instance side to the type side.
- Composite enables the WfMS to dynamically change the job structure.

2.3 Long, slow running jobs and persistence

An object-oriented system is well-suited for long lived, slow running jobs. Objects contain behavior and state and it is not difficult to make them persistent.

Long running activities are exposed to hardware and software failures. Therefore, a persistence mechanism is an important dimension of the workflow management facility. The RFPs unanimously suggest object-oriented solutions. However, they discuss persistence briefly and focus mostly on the availability of the CORBA persistence service POS [OHE97].

There are no design details. How does the facility use the persistence service? Are there any subtle points the implementers of the facility need to be aware of? Fortunately, the object-oriented community has recognized this as a recurring problem and documented it in pattern form. The Memento pattern [GHJV95] provides a design solution for capturing and externalizing an object's state without breaking encapsulation. It supplies the link between the objects in the work and flow tiers and the persistence services offered by the underlying platform.

Auditing tools (Section 2.4) may also use the Memento pattern to store audit trail information. Patterns that document techniques for recording the history of domain objects are available as well [And98].

2.4 Feedback

Job and resource management is not a small task. Workflow management systems are among the critical success factors for process-centered organizations [JEJ94]. Therefore, the ability to obtain feedback about the processes within the enterprise is essential. Measuring how the business works also enables managers to adapt to changing environments. Measurements are required for business process reenginering (BPR) and continuous process improvement (CPI) activities.

Monitoring and auditing facilities are mandatory requirements of the workflow RFP. Consequently, both submissions address this issue. In both cases, the operations that provide runtime information are implemented around the process model—Figures 1 and 2. The task controllers from the Nortel RFP contain monitoring as well as auditing operations. These provide information about the controller's state, inputs and outputs. In contrast, the jFlow RFP treats monitoring and auditing separately. Monitoring employs the status inquiry operations provided by `WfProcess` and `WfActivity` objects. A different object, `WfEventAudit`, provides audit records of workflow event information.

The patterns at the core of our job model determine how it provides information for monitoring and auditing. The type side contains information about all instances of a job type. Every job also provides instance-specific runtime information. The Type Object pattern makes it easy to localize these operations. Additionally, Composite ensures a common interface between jobs and job steps. This simplifies building monitoring and auditing tools.

3 Open Questions and Suggestions

We begin this section with a few issues that are not clearly addressed by the current RFPs. These questions will probably be answered in the forthcoming versions of these documents.

- The proposals do not specify how to define workflows. It is likely that the RFPs assume the use of the Workflow Process Definition Language (WPDL) defined by the Workflow Management Coalition. However, neither proposal mentions WPDL, although they should. In this case, the facility initializes the type side of the system from a WPDL representation. This option has the advantage that process definitions can be exchanged between WfMS and BPMTs.

  Another possibility is to define the workflow in terms of objects within the domain model. In [MJ98] we suggest five types of building blocks for a process model: primitive, parallel, sequence, alternation and repetition.

- Some of the design decisions are not properly explained. For example, the Nortel proposal identifies four types of tasks: simple, compound, genesis and adapter. It is not clear why the developers have chosen a separate class for adapter tasks. Couldn't simple tasks wrap legacy applications as well?

  The jFlow process model is similar to the job model presented in Section 2. `WfProcess` is located on the instance side, while `WfProcessMgr` is on the type side. A `WfProcess` contains a set of `WfActivity` objects. `WfActivity` does not have a correspondent on the type side. Therefore, it is likely that introducing new activities requires changing the `WfProcessMgr` accordingly.

As we stated at the beginning of this paper, the design methodology adopted by the RFPs authors is not appropriate for complex object-oriented systems. Here are a few suggestions for improvement.

- The Unified Modeling Language (UML) [FS97] is becoming the standard for modeling, building and describing object-oriented systems. However, there is more to UML than diagrams. One of the important concepts is the use case [JEJ94].

  Both RFPs employ several kinds of UML diagrams to describe the facility. Unfortunately, they do not contain any use cases, but they should. First, use cases describe how the facility interacts with its users. Second, testing begins from use cases. And finally, use cases establish a starting point for a comparative evaluation of the proposals. Ultimately, providing use cases is a way of publishing requirements. This is particularly important for proposals that aim at becoming standards for the Object Management Architecture.

- The interfaces defined by the RFP provide reusable designs. However, the documents put little emphasis on design. Potential design flaws are difficult to identify, particularly in the absence of use cases. Using patterns to document the facility helps discover design flaws early on. Patterns have proven useful in documenting large, complex object-oriented frameworks [Joh92].

- The jFlow submission misuses the term “pattern:”

  > We use standard patterns to represent attributes and relationships of the workflow interfaces. [. . .] The pattern for access operations on attributes is the following: for an attribute with name `ATTR` and type `TYPE`, two operations are provided; `TYPE ATTR();` returns the value of the attribute, `void set_ATTR(in TYPE value)` supports updates of the attribute [. . .].

  What this submission calls “patterns” are actually naming conventions. Many books on software patterns have been written since the object-oriented community adopted this idea from architecture. The term is already associated with documented solutions for recurring problems. Its misuse confirms our suspicion that the RFP’s authors are not familiar with this technique.

4 Summary

A workflow management facility is complex. Designing one is a large and difficult undertaking. Many people invested a lot of thought and time to submit the workflow RFPs.

In the previous section we have shown the Nortel and jFlow solutions, as well as a solution obtained with a different methodology. We have discussed another way to look for answers for some of the problems specific to workflow systems: changing and overriding rules, providing feedback and managing slow, long running jobs. However, with the right tools, these answers are easy to find. They have been documented and are available in the form of software patterns.

The design presented here is based on several patterns. Type Object is at the core of the job model. Its benefits are key to a dynamic system. Composite allows the facility to keep complexity under control. It also facilitates building workflows and changing their structure. Memento provides the link between domain objects and persistence services.

5 Conclusion

The RFPs submitted to the OMG consist of a set of interfaces. These interfaces provide reusable designs. Workflow management is at the cross section of several areas of computer and management science. The hard part about providing a successful design, particularly in this situation, is partitioning the functionality. The only way to get this right is to use the proper instruments and methodology.

In this paper we have focused on applying object-oriented techniques to the workflow management facility. Our starting points were the Nortel and jFflow submissions. We have discussed the solutions adopted by each submission and then showed a pattern-based solution. We have emphasized a design methodology that enables developers to craft robust, reusable object-oriented systems.

The objective of this paper is twofold. First, we want to investigate what types of patterns are applicable to workflow. The previous sections show that the core of the facility can be expressed as a combination of a few patterns. Our quest is not over, though. We will continue to seek and document patterns for workflow systems. Second, we would like to share with the workflow community a better approach to object-oriented design. The increasing number of large projects that embrace it demonstrate its efficiency. We are looking forward to disseminating and applying these techniques.

Ideally, the RFPs would be documented with a set of patterns. Such designs take less time to complete. They are shorter and easier to understand. They are also reusable. The time saved could be used to justify the design choices and provide a solid set of use cases. To end with the words of Richard Gabriel [Gab96]:

> The promise of object-oriented programming—and of programming languages themselves—has yet to be fulfilled. [. . .] To some degree, this failure can be attributed to a failure of the design methodologies we have used to guide our design of languages, and to a larger degree to our failure to take seriously the needs of the programmer and maintainer in caretaking the code for a large system over its life cycle.

References

[AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and limitations of current workflow management systems, 1997. Available on the Web at `http://www.almaden.ibm.com/cs/exotica/wfmsys.ps`.

[And98] Francis Anderson. A collection of history patterns. In *Proc. 5th Pattern Languages of Programming*, Monticello, Illinois, August 1998. Available on the Web at `http://st-www.cs.uiuc.edu/~plop/plop98/`.

[BMR$^+$96] Frank Buschman, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture—A System of Patterns*. John Wiley & Sons, July 1996.

[DKOS98] A. Dogac, L. Kalinichenko, T. Ozsu, and A. Sheth, editors. *Advances in Workflow Systems and Interoperability*. Springer-Verlag, 1998. To be published.

[FS97] Martin Fowler and Kendall Scott. *UML Distilled—Applying the Standard Object Modeling Language*. Object Technology Series. Addison-Wesley, June 1997.

[Gab96] Richard P. Gabriel. *Patterns of Software—Tales from the Software Community*. Oxford University Press, 1996.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns—Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GHS95] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases, an International Journal*, 3:119–153, 1995. Available on the Web at `ftp://ftp.gte.com/pub/dom/reports/GEOR95a.ps`.

[GT98] Dimitrios Georgakopoulos and Aphrodite Tsalgatidou. *Technology and Tools for Comprehensive Business Process Lifecycle Management*. In Dogac et al. [DKOS98], 1998. To be published.

[JEJ94] Ivar Jacobson, Maria Ericsson, and Agneta Jacobson. *The Object Advantage–Business Process Reengineering with Object Technology*. Addison-Wesley, 1994.

[JGJ97] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse—Architecture, Process and Organization for Business Success*. ACM Press Books. Addison-Wesley, 1997.

[Joh92] Ralph E. Johnson. Documenting frameworks using patterns. *ACM SIGPLAN Notices*, 27(10):63–76, October 1992. *OOPSLA '92 Proceedings*, Andreas Paepcke (editor).

[Jol95] David Jollingsworth. *The Workflow Reference Model*. Workflow Management Coalition, Avenue Marcel Thiry 204, 1200 Brussels, Belgium, 1995. Available on the Web at `http://http://www.aiim.org/wfmc/`.

[JW97] Ralph E. Johnson and Bobby Woolf. *The Type Object Pattern*, chapter 4. In Martin et al. [MRB97], October 1997.

[MJ98] Dragoş-Anton Manolescu and Ralph E. Johnson. A proposal for a common infrastructure for process and product models. In *OOPSLA Mid-year Workshop on Applied Object Technology for Implementing Lifecycle Process and Product Models*, Denver, Colorado, July 1998. Available on the Web at `http://www.uiuc.edu/ph/www/manolesc/oopsla/`.

[Moh97] C. Mohan. Recent trends in workflow management products, standards and research. In *Proc. NATO Advanced Study Institute (ASI) on Workflow Management Systems and Interoperability*, Istanbul, Turkey, August 1997. Springer-Verlag. Available on the Web at `http://www.almaden.ibm.com/cs/exotica/wfnato97.ps`.

[MRB97] Robert C. Martin, Dirk Riehle, and Frank Buschmann, editors. *Pattern Languages of Program Design 3*. Software Patterns Series. Addison-Wesley, October 1997.

[OHE97] Robert Orfali, Dan Harkey, and Jeri Edwards. *Instant CORBA*. John Wiley & Sons, 1997.

[OMG97] Workflow management facility request for proposal. OMG Document cf/97–05–06, May 1997.

[OMG98a] Joint workflow management facility—revised submission. OMG Document Number bom/98–06–07, 1998. Available on the Web at `ftp://ftp.omg.org/pub/docs/bom/98-06-07.pdf`.

[OMG98b] Workflow management facility specification. OMG Document Number bom/98–03–01, 1998. Available on the Web at `ftp://ftp.omg.org/pub/docs/bom/98-03-01.pdf`.