An Extensible Workflow Architecture with Objects and Patterns

Dragos A. Manolescu
micro-workflow.com and the University of Kansas
`dragos.manolescu@acm.org`

Abstract

Despite the large number of commercial workflow systems, object-oriented developers implement their business processes with home-made workflow solutions. Current workflow architectures are based on requirements and assumptions that don't hold in the context of object-oriented software development. Micro-workflow, a new workflow architecture, bridges the gap between the type of functionality provided by existing workflow systems and the type of workflow functionality required by developers. Micro-workflow provides a better solution when the focus is on customizing the workflow features and integrating with other systems.

1 Introduction

Workflow technology has been used for decades to automate various "processes."[1] In the 1970s, Hammer and colleagues [12], Zisman [25], and others focused on procedures for Office Information Systems. Around the same time, Ellis and Nutt [8] recognized the importance of developing models of office procedures while working on Officetalk, an experimental office information system at Xerox PARC. Research during the 1980s placed more emphasis on process models. Winograd and Flores proposed a speech act model [18] while Ellis and Nutt worked on Petri net models [9]. Workflow expanded into fields like office automation and document imaging.

During the last few years workflow has been the focus of intense activity in terms of products, standards, and research work. It is no longer just "some sort of planned document routing" [24]. Currently workflow technology and process support lies at the center of modern information systems architectures. Consequently an increasing number of developers are embracing this technology. This trend has pushed workflow from an end-user application into middleware services. OMG's adoption of a workflow management facility [22] provides a clear sign of this shift.

Software developers want workflow systems that they can integrate with and tailor for their applications. However, current workflow systems are based on assumptions and requirements that don't hold in the context of software development. They target non-programmers and therefore package a wide range of features. This focus produced heavyweight and monolithic architectures, which are hard to customize and extend. This mismatch forces developers to build home-made workflow solutions whenever they need workflow functionality within their applications.

This paper presents a new workflow architecture that solves the above mismatch. Section 2 provides three motivating examples, and Section 3 analyzes the problem. Section 4 introduces my solution, the micro-workflow architecture. Then Section 5 evaluates the solution, and Section 6 draws conclusions and discusses future work.

[1]Workflow is applicable to many other domains besides the business domain. For simplicity, this paper uses the term "process" instead of "business process," "administrative process," "scientific process," etc.

2 Motivation

Many workflow researchers have stumbled over various limitations of traditional workflow architectures that contribute to making them unsuitable for implementing processes within object-oriented applications. Here are some notable examples related to the subject of this paper.

The experience with the Mentor workflow system helped researchers at the University of Saarland realize the limitations of current monolithic, heavyweight workflow architectures. Consequenly they have proposed "a review of current architectures of workflow management systems" [20]. Currently they are developing Mentor-lite, a new generation workflow system centered around a lightweight core. Unlike its predecessor, Mentor-lite provides advanced workflow features like worklist management, history management, and monitoring as *extensions*, implemented as workflows on top of this core.

The OPERA project at ETH Zürich started from similar findings [2]. First, the narrow purpose design of current workflow systems limits their applicability to the domain and applications for which they have been tailored. Second, monolithic workflow architectures lack the capabilities required to adapt them to new kinds of applications. OPERA proposes a *kernel* for process management where features like inter-process communication are implemented by separate *components*.

Other research efforts in the workflow domain have also uncovered weaknesses in the reference models adopted by the Workflow Management Coalition (WfMC) and the OMG. For example, Paul and colleagues [23] point out that the since the WfMC Reference Model resembles the architectures of current workflow systems, *it inherits their problems*. The monolithic server of the WfMC standard impedes the flexibility and scalability of workflow systems, and makes them unsuitable for Internet-wide workflow applications.

Christoph Bussler discusses the challenges of enterprise-wide workflow [4]. He finds that current workflow management systems fall short in important areas required for building and maintaining reliable workflow infrastructures. These problems are caused by current workflow architectures providing functionality that belongs to other subsystems. He suggests *normalization by componentization* as a possible solution, with many workflow systems sharing components which implement common functionality.

Let's consider several applications that illustrate the mismatch between the type of functionality provided by traditional workflow architectures and the one required within object-oriented applications. Each application implements a real-world process and has different workflow requirements. My dissertation provides detailed descriptions of these processes [14].

2.1 NCSA Proposal Review

This application implements an administrative process at the National Center for Supercomputing Applications (NCSA). NCSA owns several supercomputers. To request CPU time on an NCSA machine, potential users submit proposals to the NCSA Allocations Office. Several reviewers study each proposal and decide whether to grant the request or not. Sometimes the reviewers also adjust the total amount of CPU time requested.

This application requires *basic workflow functionality*, and the ability to *monitor* and *record* process execution. Another characteristic that makes this example interesting is that it requires workflow functionality that *integrates with application objects* which provide their own interfaces for human workers.

Can current workflow systems implement this process? They can, but this is not the right question to ask. The question should rather be: can developers use the functionality provided by an existing workflow system to build an application implementing this process? Because most workflow systems focus on packaging many features, the cost of a solution based on one of these systems would be much higher. Developers shouldn't pay for features they don't need. Nor should they have to bundle unused features in their applications. Unfortunately, due to the monolithic nature of traditional workflow architectures, current workflow systems can only be used in an all-or-nothing manner. This approach makes it hard to reuse the functionality provided by current workflow systems within object-oriented applications. Additionally, these systems don't support the incremental integration of workflow within applications, and they don't let developers customize their features.

2.2 Strep Throat Treatment

This application implements a process from the medical domain. In addition to *basic workflow functionality*, this process requires: *recording the workflow history to a persistent store* for legal reasons; allowing the physician to *change a running workflow*; and *supporting human workers*. Therefore this application requires a workflow system that supports manual intervention, persistence, and worklists.

Although various studies have identified the absence of support for manual intervention as one of the shortcomings of current workflow systems [1], only few of them have this feature. The Strep Throat Treatment Process can be implemented with a workflow system only if this supports manual intervention. However, a solution built with a traditional workflow architecture doesn't let developers to pick and choose the workflow features this application requires. Additionally, they have no or very limited control over the features bundled with the system.

2.3 Newborn Followup

This application implements an administrative process from the Newborn Screening Program at the Illinois Department of Public Health (IDPH). Hospitals throughout the state collect blood samples from newborn babies and send them to a Chicago-based laboratory for testing. The followup process is triggered only when the lab returns anomalous test results. The objective of the process is to keep track of these problems, and ensure that they are dealt with according to the state law.

The IDPH Newborn Followup Process requires a workflow system providing *basic workflow functionality* and support for *federated workflow*—a workflow fires off one or several remote subworkflows. Each instance of this process involves two workflow systems, one at each field office and one at the lab. Currently most workflow systems don't have the ability to integrate multiple workflows distributed over the Internet. Additionally, in the absence of a common standard, the integration of heterogeneous workflow systems requires access to their internals. But the black-box design of current workflow systems makes them hard to integrate.

3 The Problem

The three applications discussed in Sections 2.1–2.3 have workflow requirements. It makes sense for their developers to seek tried solutions that aim at providing process support—workflow technology.

In general, to accommodate the requirements of applications that implement processes, developers need a workflow system that: (1) Allows them to pick and choose the workflow features required by each application; (2) Lets them customize existing features and add new ones; (3) Integrates with custom components, subsystems, and frameworks; and (4) Supports incremental integration within existing domain objects and applications.

But current workflow systems don't fit this bill. They are incompatible with these requirements. Their monolithic, heavyweight architectures have been designed under different assumptions. Solving this mismatch requires a new generation of workflow architectures. Let's look now at a solution that revolves around objects and patterns.

4 The Micro-Workflow Architecture

To bridge the gap between the type of workflow functionality provided by products currently available on the market and the type of workflow functionality developers need to implement processes within object-oriented applications I have developed micro-workflow, a new workflow architecture. I have implemented the architecture as an object-oriented framework in VisualWorks Smalltalk, on Linux systems.

Micro-workflow regards object-orientation as an architectural style. The object paradigm requires finding abstractions for the problem domain, partitioning the functionality, and defining interfaces. These are all hard problems, but when successfully solved they could yield powerful, reusable systems [13]. Many programmers can decompose a problem into a set of classes. What differentiates experts from novices is that the former craft their objects such that they can leverage all three characteristics of object systems—encapsulation, inheritance, and polymorphism. This requires a sound understanding of the problem domain, anticipating how requirements will change, as well as mastering the techniques of good object-oriented design.

At first sight, the object paradigm doesn't seem appropriate for workflow management. Object-oriented systems lack a procedural representation of control flow. The decomposition into classes typical of object-oriented architectures deemphasizes the control flow, distributing it among different objects. Thus, the global control flow and behavior are less visible than in procedural programs. Therefore, an additional challenge of building an object-oriented workflow architecture lies in providing abstractions that maintain an explicit representation of the control flow without violating the principles of good object-oriented design.

4.1 Object-Oriented Workflow Architecture

An object-oriented workflow architecture must *provide abstractions* that enable software developers to define and enact how the work flows through the system. It should also allow them to *tailor the architecture* in ways specific to object systems. Object-oriented developers use several techniques to customize object systems.

White-box techniques involve specializing objects through subclassing. For example, developers should be able to add a new workflow control structure by subclassing an abstract class. Black-box techniques involve tailoring the functionality through aggregation. For example, developers should be able to add a workflow-specific feature by plugging in an object providing the desired functionality.

There are many workflow systems built around objects. However, they don't fully exploit the potential of object technology; rather, they regard it mainly as an implementation technique [19]. So why don't workflow architectures fully embrace object technology? Unfortunately, since traditional workflow architectures target end-users, they tend to focus on the number of features instead of reuse and flexibility. Moreover, most people who build workflow systems are just starting to discover the techniques that enable them to use objects efficiently. The micro-workflow architecture leverages all three characteristics of object systems to foster reuse and flexibility.

4.2 Component-based Workflow Architecture

Micro-workflow encapsulates workflow features in separate components. At the core of the architecture, several components provide basic workflow functionality. Additional components implement advanced workflow features. Figure 1 illustrates these components.

Through composition software developers extend the core only with the features they need. This design localizes the changes required to tailor a workflow feature to the component that implements it. Additionally, developers can add new features by building new components. In effect, this architecture follows the *Microkernel* pattern [3] (thus the prefix "micro-").

4.3 The Micro-Workflow Core

The micro-workflow architecture adopts a minimalistic approach. At the core of the architecture, three separate components allow developers to define and execute workflows (i.e., provide basic workflow functionality):

- The *excution component* provides the mechanism that executes workflows based on their definition. This mechanism parallels an object model and is implemented around the *Type Object* pattern [17].

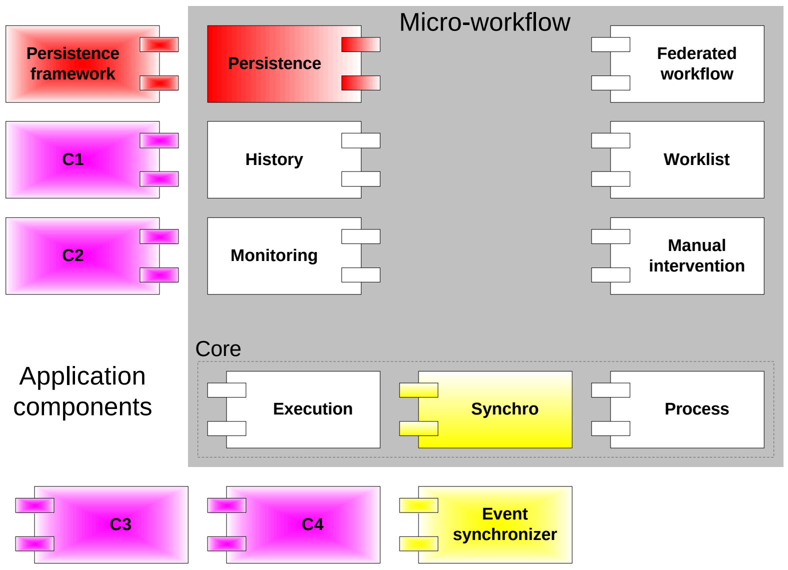

Figure 1. The micro-workflow architecture. The gray box contains the micro-workflow components. Other application components reside outside this box.

- The *process component* is based on an activity-based process model. It provides the abstractions required to build workflows. Developers define processes by creating, configuring, and connecting objects. The implementation relies on the properties of the *Composite* pattern [10] to allow hierarchical decomposition.
- The *synchronization component* uses Event-Condition-Action (ECA) rules [5] to allow developers to define dependencies within the workflow domain.

The simplified UML class diagram from Figure 2 shows the structure of the micro-workflow core.

This approach has several advantages over traditional workflow architectures. First, its simplicity makes it easier to understand than current monolithic, heavyweight architectures. Second, each component encapsulates a design decision. This facilitates customizing or replacing individual components. For example, developers can extend the process component (one of the characteristics of a successful workflow architecture [4] which is missing from current workflow systems) through subclassing, without affecting other components. Finally, the separation of concerns simplifies integration with custom components.

One of the key features that sets the micro-workflow architecture apart from traditional workflow architectures is that the former allows developers to add advanced workflow features by adding components. The next sections extend the core with pluggable components that implement the features required by the applications from Section 3.

4.4 Monitoring Component

Workflow monitoring represents a feature common to many workflow management systems. A monitor provides information about the status of running workflows. It allows workflow users to check how the process is running, and helps them identify of out-of-line situations. Since typically workflows run for a long time, the possibility of obtaining runtime information in real time and the early identification of potential problems could save time and other resources.

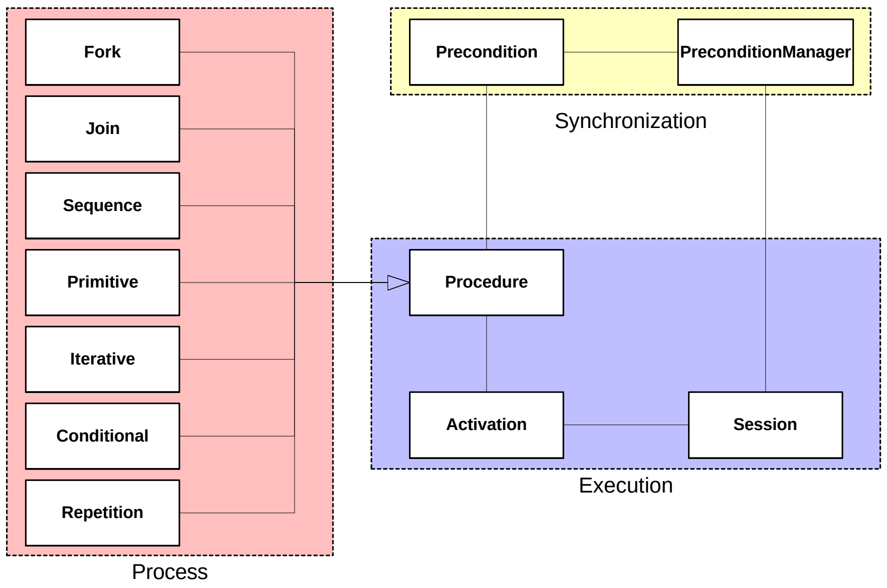

Figure 2. The micro-workflow core, simplified UML class diagram.

Most commercial workflow systems implement workflow monitoring through GUIs that provide a snapshot of what's going on in the system. However, their users have little control over what type of information these GUIs display, and typically can't use this data for other purposes.

I have implemented the micro-workflow monitoring component around the *Observer* pattern [10]. This design separates *how* micro-workflow tracks workflow execution from *what* it does with this information. As an example of the latter the micro-workflow framework uses the ProcedureMonitorInterface, a GUI that resembles the view provided by a debugger. However developers can use the runtime information extracted by the ProcedureMonitor in other ways.

My approach lets developers add monitoring to the framework *only when their applications need this functionality*. The design localizes the changes required to customize this feature, and facilitates integration within existing application. For example, many applications already have a GUI that displays status information. Additionally, it lets developers use the monitoring data for other purposes besides presentation. For example, this data could fire off triggers that signal abnormal situations.

4.5 History Component

Workflow management systems log the execution history of the workflows they execute. There are two reasons to collect history information. First, *workflow users* may use it after the workflow completes execution. For example, process designers use it to evaluate, improve, and even derive new process models; auditors use it for auditing purposes; etc. Second, the *workflow system* may also use the history information for recovery purposes.

Although most commercial workflow products log process execution, they provide limited access to the history mechanism. Typically their users have little control over *what* information the system records, and no control over *how* and *where* the system stores this information. This approach works well for non-technical people. End users don't care how the history mechanism works and are not interested in changing it. However, developers who use workflow to implement processes within applications want to be able to tailor the history mechanism, and customize it for particular problems. Therefore, a workflow architecture targeting software developers should provide fine grained access to and control of the history mechanism.

For this first application I have implemented the micro-workflow history component around the *Singleton* pattern [10]. This component extracts the runtime workflow data from the execution component and logs it in memory, through the sole instance of a Logger class.

Software developers using the micro-workflow framework plug in this component *only when their applications need history*. The compositional approach lets them customize what workflow data is logged into the workflow history while reducing the impact over the execution component.

4.6 Persistence

In traditional workflow architectures, history encompasses choosing the type of information the system records about workflow events, as well as specifying *where* the system stores this information. Since non-technical people use workflow systems without requiring access to these mechanisms, treating them together is a valid design decision.

Current workflow systems don't discriminate between history and persistence. They record the runtime data (i.e., workflow history) into a persistent store. Most of them rely on relational database technology.

Micro-workflow regards workflow history and persistence as orthogonal features. Unlike traditional workflow architectures, it separates *extracting* the workflow event data (which is the responsibility of the history component) from *storing it into a persistent store*. A separate persistence component provides the access point to a database. I have implemented the persistence component with the GemStone/S object-oriented database.

Software developers using the micro-workflow framework plug in this component *only when their applications need persistence*. The ability to use this component only when needed is in line with Klaus Hagen's findings; his PhD thesis concludes that in workflow management systems "there should be the possibility to control whether a process is made persistent or not" [11]. Through localizing the database-dependent part within the persistence component, my design also provides *database independence*. Developers can replace the underlying database without affecting other framework components. They can even use an object-oriented database, or (provided an appropriate object-to-relational layer) a relational database. Notice, however, that adding this component also required refactoring the history component. I have redesigned the history component around the *Strategy* pattern [10]. Currently the framework provides logging strategies that let developers discard the workflow history, log it in memory or, with the help of the persistence component, log it to a persistent store.

4.7 Manual Intervention

Workflow management systems must allow authorized users to change the sequencing of activities while the workflow is running. If they don't, their users perceive them as too rigid and avoid using them altogether [21]. In the case of an activity-based process model, this corresponds to manually moving the flow of control from one node of the activity map to another.

Several studies identify the support for manual intervention as one of the important requirements of workflow systems. Despite their findings, currently only few commercial systems provide this feature [1].

The manual intervention component provides the mechanism for stopping, rewinding, and resuming running workflows. In effect, it lets workflow execution jump to any point within the process history—therefore it depends on the history component described at the beginning of Section 4. Instances of a Rewinder class encapsulate this functionality. However, altering process execution at run time translates into manipulating the workflow call stack. Therefore this component requires separate control flow mechanisms for the *implemented* system (micro-workflow) and the *implementing* system (Smalltalk).

The compositional approach adopted by the micro-workflow architecture lets developers plug in this component *only when their applications require support for manual intervention*. The separation of concerns provides full control over the backward recovery mechanism. Changing the way the framework handles the cancellation of already executed activities as it backtracks through the history impacts only the Rewinder class.

4.8 Worklist Component

Micro-workflow processes involve workflow objects which encapsulate the process logic, and application objects which encapsulate the task logic. However, workflows typically involve human workers as well as application objects. In fact, the synergy between humans and software represents one of the key features of workflow. Therefore, an object-oriented workflow management system must accommodate human workers as well as application objects.

Worklist management is an integral part of current workflow architectures. This makes them unusable for applications that provide their own worklist management and therefore don't need this functionality (e.g., the application discussed in Section 2.1). Some workflow systems also deal with the organizational dimension. This means that the part of the system implementing worklists also handles staff resolution, in addition to the invocation mechanism and interfaces for human workers. Micro-workflow regards the organizational issues as beyond the responsibilities of a workflow system. This philosophy keeps the architecture lightweight, which is one of the main goals of micro-workflow.

The micro-workflow architecture relies on a separate component to provide the human-computer interface and the asynchronous invocation mechanism required to support human workers. This approach decouples the workflow core from the issues of human-computer interaction, supporting the idea of extending the core functionality through composition. It also relieves the workflow core from the job of figuring out how to reach its users and send out their workitems. The micro-workflow worklist component uses future objects, which I have implemented with classes from the Smalltalk class library and reflection. My dissertation provides complete implementation details [14].

Micro-workflow lets software developers plug in the worklist component *only if their applications requires its functionality*. This approach is consistent with one of the main goals of the architecture: add features by adding components. Additional benefits of this solution include customized worklist handling and integration with existing directory services. Notice that in addition to supporting human workers, the worklist mechanism allows for other workflow features that require asynchronous invocation. For example, disconnected operation lets workflow workers remove their laptop computers from the network while they process their worklists [19].

4.9 Federated Workflow Component

The federated workflow component extends the micro-workflow core with support for *hierarchical* and *distributed* workflow [15] (Figure 3). In effect, this component allows for transparent workflow distribution across the enterprise. This lets software developers depart from the centralized model of workflow execution, and implement workflows that execute parts of the process at multiple locations.

The ubiquity of computers and the Internet have made federated workflow a powerful feature, particularly with the increasing popularity of outsourcing. However, federated workflow is currently not available in most workflow management systems.

Hierarchical workflow translates into the ability to have other workflow systems implement activity nodes of the process definition. I have introduced two new abstractions for this purpose. Workflow represents a workflow system. SubworkflowProcedure is a new control structure that executes another Workflow—i.e., a subworkflow.

However, the Workflow and SubworkflowProcedure deal only with hierarchical workflow. They assume that the parent workflow and the subworkflow reside within the same address space. Distributed workflow provides the ability to execute workflows regardless of their location. Following the *Facade* pattern [10], I have introduced a WorkflowFacade to represent an abstraction for remote workflow systems. The WorkflowFacade class adds support for *distributed workflow* by leveraging the infrastructure provided by the VisualWorks Opentalk [6], a CORBA-based distributed application environment. This enables a workflow running within one address space (i.e., the server) to execute a subworkflow within a different address space (i.e., the client). Each WorkflowFacade instance uses the Opentalk naming service to obtain a reference to its peer.

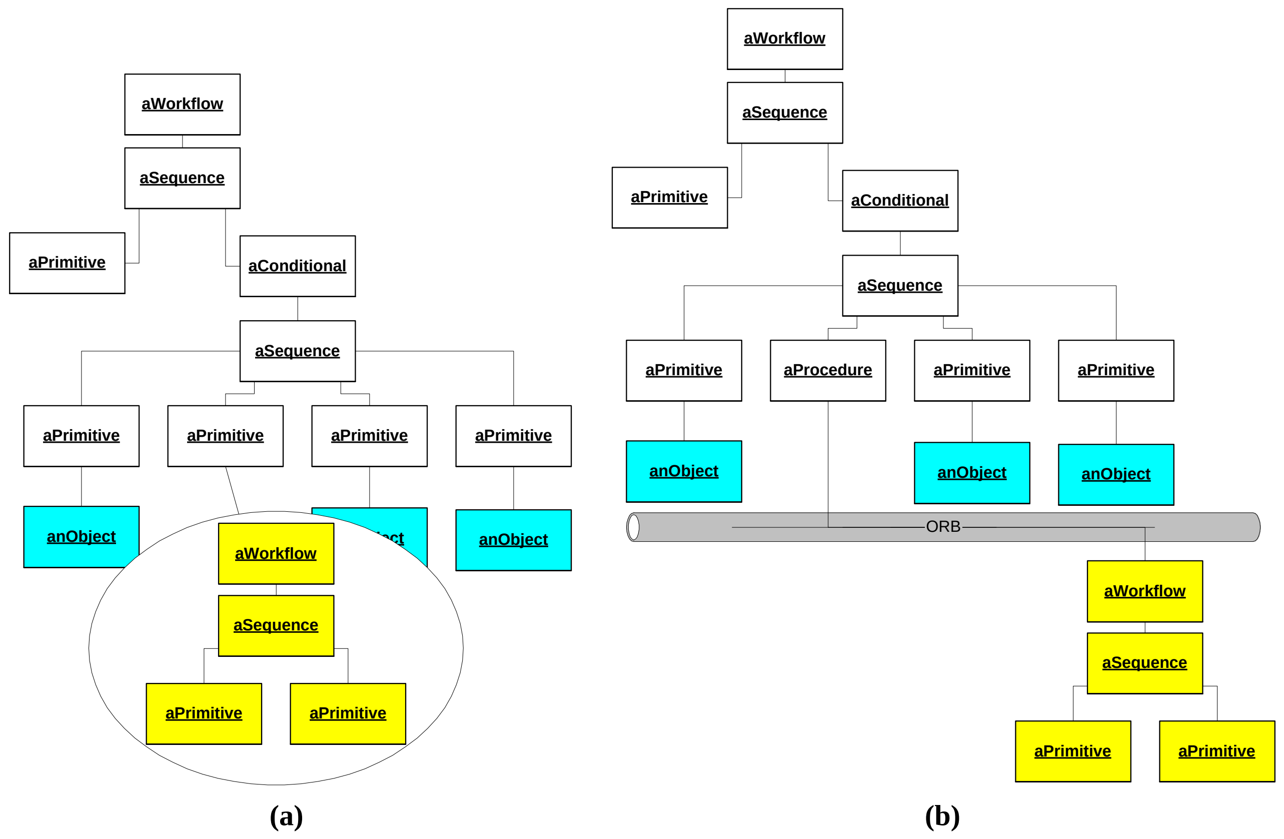

Figure 3. Federated workflow corresponds to hierarchical workflow (a) and distributed workflow (b)—simplified UML instance diagrams. The distributed workflow configuration shown here uses an ORB, but the design lets developers use other distributed application architectures.

This design has several important characteristics. First, developers add the federated workflow component *only when their applications require breaking up workflow execution and distributing it among different workflow systems*. Second, the federated workflow component described in this section assumes a homogeneous federation. However, since the abstractions aim at offering a consistent view *above* the Workflow class (i.e., this class decouples clients and providers), this assumption *doesn't affect* the framework—the glue code connecting heterogeneous workflow systems is not reusable anyways. Additionally, having the same system implement both the workflow and the subworkflow reveals the issues that have to be dealt with at both ends. And finally, the WorkflowFacade class hides the details about the remote invocation of other workflow systems behind an invariant interface. This lets developers to change the underlying distributed application architecture (e.g., go from a CORBA-centric solution to one based on DCOM or RMI).

4.10 Extending the Architecture

The micro-workflow components introduced in this section implement a wide variety of workflow features, ranging from history to federated workflow. The compositional approach adopted by the micro-workflow architecture benefits software developers who need workflow functionality within their applications in several ways. First, the ability to add features by plugging in components enables them to tailor the workflow functionality to their requirements. Second, developers could customize each workflow feature individually. The compositional design localizes most changes required to modify a feature within the component that implements it. Third, the architecture can grow and provide new features. Developers add new features through building new components and adding hooks for them within the core. Finally, having a component encapsulate each feature facilitates integration with custom components.

Table 1 summarizes how developers extend the micro-workflow core with the components discussed in this section. For example, to implement a workflow that requires only history and worklists, developers add the corresponding components to the micro-workflow core. Adding the history component involves plugging a logging strategy instance into the workflow session. Adding the worklist component involves initializing the workflow context with a Worklist instance for each workflow actor; provide the code that handles work item removal and returning results; and specializing Workitem to display the work items in a manner appropriate for the human workers.

Component	**How to add to the core**
History	Plug an instance of a concrete subclass of LoggingStrategy into the workflow session
Persistence	Select a logging strategy that uses a persistent store; plug in a SessionManager instance and execute the workflow within a database session
Monitoring	Register a ProcedureMonitor instance as a dependent of the workflow session
Manual intervention	Supply the workflow session with the class that implements the rewinding mechanism
Worklist	Replace domain objects within the workflow context with Worklist instances; provide the application-specific code that handles work items once they're taken off the worklist, and passes back the domain object once the user completes processing; subclass Workitem to control how the worklist displays each work item
Federated workflow	Build a process containing SubworkflowProcedure instances with WorkflowFacade instances; register the facades with the name servers; implement the Opentalk pass-by-value mechanism for objects whose instances should be passed by value

Table 1. Extending micro-workflow with advanced workflow features.

However, this section raises two important questions. First, is this approach worth the trouble? Adding new components to the framework is feasible only if it requires less work than extending a monolithic workflow system. Second, what is the run time cost of the flexibility provided by this approach? If the cost is too high, software developers won't use the architecture. The answers to these questions determine whether micro-workflow provides a viable solution for implementing workflows within object-oriented applications. The next section provides the answers.

5 Evaluation

The previous section has discussed how the micro-workflow framework evolved to accommodate the workflow requirements of the three applications introduced in Section 3. I have built these applications with the micro-workflow framework [14]. This proves that the micro-workflow architecture can be implemented with an object-oriented language (Smalltalk), an object-oriented database (GemStone/S), and a distributed application architecture (Opentalk).

But the additional flexility of the micro-workflow architecture has its costs. The *design cost* corresponds to the amount of effort required to add new features, and the breakage caused on the core components. It helps software developers estimate whether they can afford implementing a new feature. Likewise, the *run time cost* corresponds to the overhead incurred by the hooks that support pluggable components. This cost affects all applications, regardless of what micro-workflow components they use. Based on metrics collected as the framework evolved to accommodate different workflow features, my dissertation provides figures for both costs [14]. The numbers show that they are within the budget of most developers.

Notice, however, that the framework was not designed upfront for the applications presented here. Rather, I started with the core and evolved it to support monitoring, history, persistence, manual intervention, worklists, and federated workflow. Each application shaped the framework according to its requirements. In the light of the experience described in this paper, it is reasonable to assume that adding other components is likely to impact the framework in similar ways.

6 Conclusions and Future Work

This paper started with the observation that current workflow systems do not provide the workflow functionality required in object-oriented applications, so developers are forced to build custom workflow solutions. Traditional workflow architectures are based on requirements and assumptions that don't hold in the context of contemporary object-oriented software development. This mismatch makes current workflow systems unsuitable for developers who need workflow within their applications.

I have presented micro-workflow, a novel workflow architecture that resolves this mismatch. While several research projects focus on a new generation of workflow architectures, I have taken a unique approach. Micro-workflow solves workflow problems through techniques specific to object systems and compositional software reuse. It aims at software developers and provides the type of workflow functionality they need in object-oriented applications. The components at the core of the architecture provide basic workflow functionality. Other components implement advanced workflow features. Software developers select the features they need and add the corresponding components to the core through composition.

The work reported in this paper reveals several important findings. First, object technology can provide a complete architectural style for workflow systems. Data abstraction, inheritance, polymorphism, and message sends are at the foundation of the micro-workflow architecture. Second, this architecture provides a viable alternative to the heavyweight, monolithic workflow architectures. Micro-workflow shifts the focus from packaging a comprehensive set of features to keeping things simple. This facilitates integration with custom components and customization for particular applications—issues which are becoming increasingly important as workflow moves from end-user applications to middleware services. This paper also shows that the micro-workflow architecture can be built, and identifies the requirements for doing so. My framework requires an object-oriented language, reflective capabilities, and a distributed application environment. This combination yields a powerful and elegant solution.

However, the micro-workflow architecture must continue to grow. Further development requires generalizing from additional examples. I am planning to release the source code so other developers can use the framework, customize it according to their needs, and add new components. Only this type of piecemeal growth process would ensure evolution and turn it into a valuable tool.

References

[1] 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`.

[2] Gustavo Alonso, Claus Hagen, Hans-Jörg Schek, and Markus Tresch. *Towards a Platform for Distributed Application Development*, pages 195–221. Volume 164 of Doğaç et al. [7], August 1998. Available on the Web at `http://www.inf.ethz.ch/department/IS/iks/publications/ahst97b.html`.

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

[4] Christoph Bussler. Enterprise-wide workflow management. *IEEE Concurrency*, pages 32–43, July–September 1999.

[5] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Deriving active rules for workflow enactment. In *Proc. 7th International Conference on Database and Expert Systems Applications*, Lecture Notes in Computer Science, pages 94–110. Springer-Verlag, 1996.

[6] Cincom Systems, Inc. *VisualWorks Opentalk Application Developer's Guide*, 1999. Part Number P46–0131–00, Software Release 5i.1.

[7] Asuman Doğaç, Leonid Kalinichenko, M. Tamer Özsu, and Amit Sheth, editors. *Workflow Management Systems and Interoperability*, volume 164 of *NATO Advanced Science Institutes (ASI), Series F: Computer and Systems Sciences*. Springer-Verlag, August 1998.

[8] Clarence Ellis and Gary J. Nutt. Computer science and office information systems. *ACM Computing Surveys*, 12(1):27–60, March 1980.

[9] Clarence A. Ellis and Gary J. Nutt. *Modeling and Enactment of Workflow Systems*, pages 1–16. Volume 691 of Marsan [16], 1993. Invited paper.

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

[11] Claus Johannes Hagen. *A Generic Kernel for Reliable Process Support*. PhD thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, 1999.

[12] Michael Hammer, W. Gerry Howe, Vincent J. Kruskal, and Irving Wladawsky. Very high level programming language for data processing applications. *Communications of the ACM*, 20(11):832–840, November 1977.

[13] Ralph E. Johnson and Brian Foote. Designing reuseable classes. *Journal of Object-Oriented Programming*, June–July 1991.

[14] Dragoş-Anton Manolescu. *Micro-Workflow: A Workflow Architecture Supporting Compositional Object-Oriented Software Development*. PhD thesis, University of Illinois, Urbana-Champaign, October 2000. Available as Computer Science Technical Report UIUCDCS-R-2000-2186. On the Web from `http://micro-workflow.com/`.

[15] Dragoş-Anton Manolescu and Ralph E. Johnson. A micro-workflow component for federated workflow. OOPSLA2000 Workshop on Implementation and Application of Object-Oriented Workflow Management Systems III, October 2000. Available on the Web from `http://micro-workflow.com/`.

[16] Marco Ajmore Marsan, editor. *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993. Also the Proceedings of the 14th International Conference, Chicago, Illinois, USA, June 1993.

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

[18] Raúl Medina-Mora, Terry Winograd, Rodrigo Flores, and Fernando Flores. The action workflow approach to workflow management technology. In *Proc. ACM Conference on Computer Supported Cooperative Work (CSCW'92)*, Emerging technologies for cooperative work, pages 281–288, Toronto, Ontario, 1992. ACM Press.

[19] C. Mohan. *Recent trends in workflow management products, standards and research*, pages 396–409. Volume 164 of Doğaç et al. [7], August 1998. Available on the Web at `http://www.almaden.ibm.com/cs/exotica/wfnato97.ps`.

[20] Peter Muth, Jeanine Weissenfels, Michael Gillmann, and Gerhard Weikum. Mentor-lite: Integrating light-weight workflow management systems within business environments (extended abstract), October 1998. Available on the Web from `http://www-dbs.cs.uni-sb.de/~mlite/`.

[21] Gary J. Nutt. The evolution toward flexible workflow systems. *Distributed Systems Engineering*, 3(4):276–294, December 1996.

[22] 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`.

[23] Santanu Paul, Edwin Park, and Jarir Chaar. RainMan: A workflow system for the Internet. In USENIX, editor, *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8–11, 1997*, pages 159–170, Berkeley, CA, USA, 1997. USENIX.

[24] Charles Petrie and Sunil Sarin. Controlling the flow. *IEEE Internet Computing*, 4(3):34–36, May–June 2000.

[25] M.D. Zisman. *Representation, Specification and Automation of Office Procedures*. PhD thesis, University of Pennsylvania, Warton School of Business, 1977.