Aspect-Based Workflow Evolution

Boris Bachmendo and Rainer Unland

Department of Mathematics and Computer Science
University of Essen, D - 45117 Essen
{bachmendo, unlandR}@cs.uni-essen.de

Abstract. In this position paper we propose an approach for the flexible evolution of object oriented workflow implementations using AOP. We show how reusable aspects can apply changes (e.g. insertion of activities or control flow constructs) to OMG compliant implemented processes. Besides aspects providing different workflow auditing methods can trigger necessary alternations. In that way a cyclic workflow improvement can be realized.

1. Introduction

Aspect-Oriented Programming is a new software engineering paradigm which supports a separation of concerns. Concern composition is realized by extending programming language with special constructs: joinpoints (which define relevant points for the insertion of concern-related code in the application class structure), pointcuts (which describe interactions between joinpoints) and pointcut-methods (also known as advises, define action to be performed before, after or instead the invocation a certain pointcut is activated by) [6].

Different perspectives of workflow modelling and implementation, e.g. control flow (execution order), data flow (data interchange) or resources are described in [7]. The applicability of AOP for supporting flexible workflow execution was first identified in [12], that propose implementing these perspective separately using AOP and weaving them together in a workflow application.

Although workflow management arose from automating well structured repetitive production processes, the need for supporting dynamic altering workflows, e.g. in office and scientific areas, is obvious nowadays. [3] distinguish between *static workflow evolution*, i.e. modifying workflow models, and *dynamic evolutio*n, i.e. adapting running process instances. Unlike [12] we propose dynamic evolution of existing object oriented workflow implementations by weaving appropriate aspects. This approach allows the reuse of both adaptation cases realized as aspects (e.g. insertion of control flow constructs) and workflow implementations. At the same time aspects implementing arbitrary auditing methods can be used to control process execution and trigger workflow evolution when necessary.

In the next section we briefly present an object-oriented workflow implementation approach. Possible evolution scenarios of control flow as well as resource and auditing perspective are described in the third section. Section four summarizes the paper and discusses some open issues.

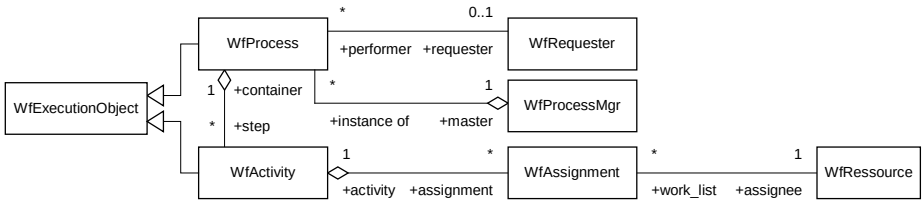

Fig. 1. Simplified Workflow Management Facility Model

2. Object-Oriented Workflow Implementation

First workflow management systems (WfMS) had a highly centralized architecture with a single workflow engine (e.g. FlowMark) or multiple replicated execution units (e.g. MOBILE [7]). But an optimal level of flexibility and scalability can only be provided by a truly distributed object-oriented implementation, where workflows are realized as independent distributed objects. In order to standardize object-oriented WfMS and to make them compliant to Object Management Architecture the OMG proposed the Workflow Management Facility Specification. Since we will explain our approach using this model as a reference, we will briefly outline it in the following.

A simplified version of Workflow Management Facility Model [10] is depicted in Figure 1. `WfRequester` interface represents a request for some job to be done. `WfProcess` has to perform the work, inform its requester upon completion and deliver him the execution result. The requester uses `WfProcessMgr` to create a process instance, i.e. it is a factory and locator for the instances of certain process type. `WfExecutionObject` is a basic interface which contains common members of `WfProcess` and `WfActivity`. While a `WfProcess` implements an instance of a particular process model, the single process steps are represented by `WfActivity`. The process creates corresponding activities and defines its context (i.e. input data). The result that is produced by activity can be used to determine the following one. `WfActivity` can also act as `WfRequester`, thereby the process it creates becomes a subprocess of its owner. `WfResorce` represents a resource necessary for activity execution.

3. Workflow Adaptation Using Aspect-Oriented Programming

3.1 Control Flow Perspective

First of all we consider the control perspective and describe how the control flow can flexibly be changed using AOP. Figure 2 depicts the insertion of activities and control flow constructs, defined by the Workflow Management Coalition [14], such as sequence, split, join and iteration. Activity diagrams on the left side show a process fragment, while sequence diagrams to the right represent interactions between objects. Replaced control flows are depicted by dashed arrows in activity diagrams, while interactions contain some special constructs showing interceptions by the aspects.

In Figure 2a an activity (A1) which is an instance of `WfActivityA` is replaced by the instance of `WfActivityB` (A2). We assume `WfActivityA` and

`WfActivityB` to be subtypes of the OMG-interface `WfActivity`. In this case we use an aspect that has a pointcut activated by the invocation of an `WfActivityA`-constructor performed by `WfProcess` instance P. The corresponding pointcut method is executed *instead* of the original invocation. In the sequence diagram the original call is depicted as a dashed connector with a transparent dot at the beginning and transparent arrow at the end. Although this call is not executed it should especially be depicted for instead-invocations to clarify what pointcut was activated. The aspectual invocation is depicted by the connector between the caller object and the aspect instance with the black dot on the aspect side. It is labelled with the original method call.

In this case (Fig. 2a) the pointcut-method creates an instance of `WfActivityB` and returns it to P instead of a `WfActivityA`-object (we assume the process is handling its activities through the `WfActivity` interface). The context used for the creation of A2 can differ from the original data. A2 considers P as its owner process and reports it the execution results. Deletion of activities can be realized analogously by replacement by dummy activities.

In Figure 2b a new activity (A2) is inserted between two existing ones and all of them are executed in a sequence. The activity constructor invocation is once again intercepted by the aspect. But in contrast to the first example the pointcut method is executed *before* the constructor. It creates a new instance of `WfActivityB`. This activity cannot report its results to the process, because P is not aware of its existence and it would interpret the call as the result of A1. So the result is reported to the aspect and thereafter the intercepted constructer call is executed. The context passed over to A1 can be derived from the A2 result which in that way can influence the rest of the process.

Figure 2c shows an inserted AND-Split between the activities A0 and A1. A single thread of control now splits into two concurrently executing threads [14]: the old (starting with A1) and the new one (A2). In contrast to the previous case the aspect does not wait until A2 is finished before it continues the instantiation of A1. Therefore the context of A1 cannot be affected by A2, whose returning result is omitted since it is not relevant. An OR-Split (i.e. branching into several alternative threads) can be inserted analogously to the activity replacement with the help of an instead pointcut method. The method evaluates given conditions and decides on creating either A1 or A2 (XOR-Split) or both of them.

Multiple threads converge into a single one by using the join construct [14]. The insertion of an AND-join that merges parallel threads is depicted in the Figure 2d. Since the coordination of A0 and A1 is already handled by the process, the aspect has to ensure that A1 can only start after A2 is finished. Therefore one pointcut observes the final call of A2 that return the result and sets an internal flag as soon as it was executed. Another pointcut method intercepts the instantiation of A1 and lets it proceed only after the flag was set. If the converging branches are alternatives (OR-Join) the aspect has to detect the termination of the both, A0 and A2. It has to trigger the creation of A1 at the moment the first of these events occurs and has to prevent the instantiation when the second one takes place.

An iteration (i.e. repetitive execution of a process segment) is added in Figure 2e. Activity A1 is performed repeatedly as long as a certain condition is fulfilled. A before pointcut detecting the result delivery of A1 and starting it again and again, is

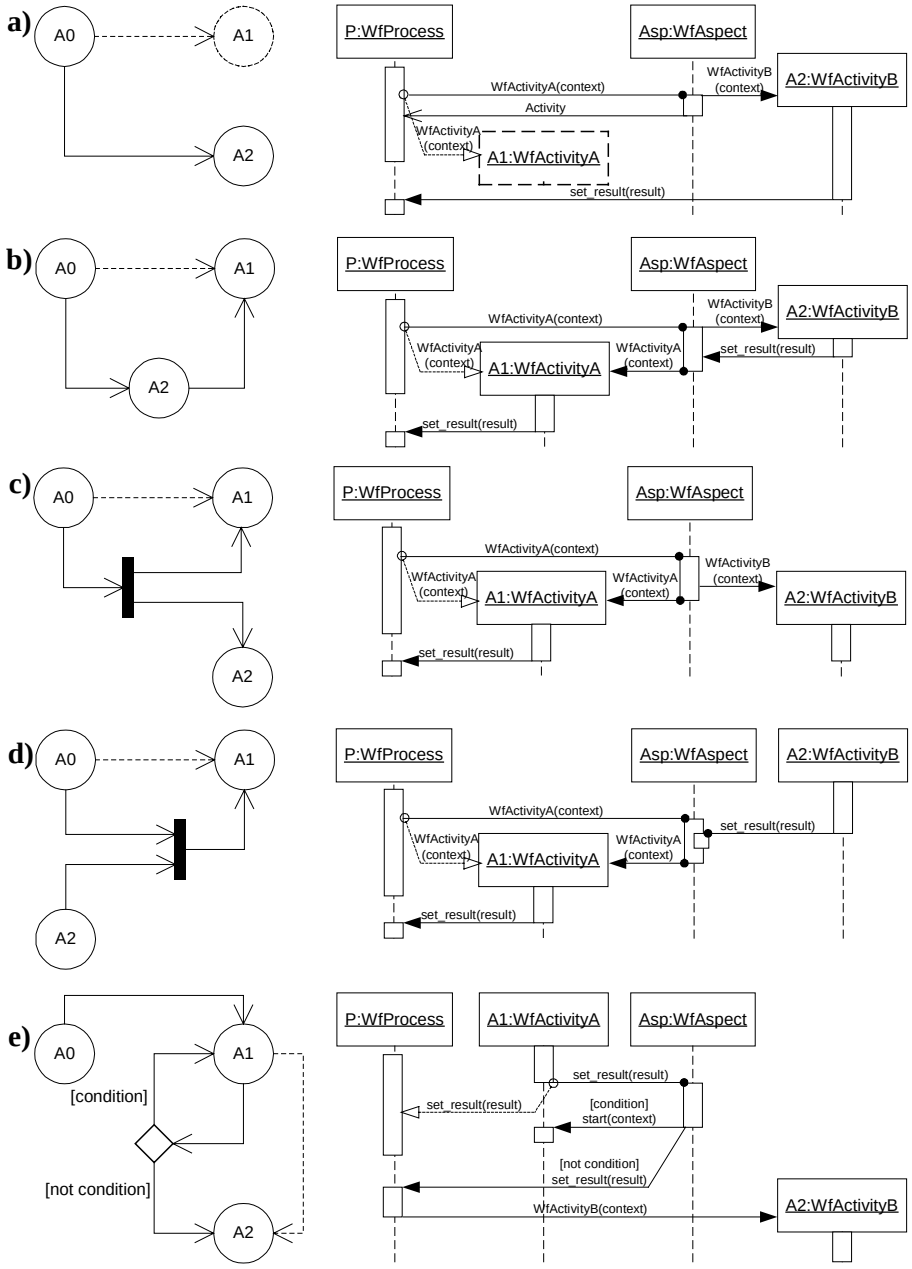

Fig. 2. Control flow adaptation using aspects.

not appropriate in this case, since it would mean that results are reported repeatedly to the process object, that cannot handle them, since it is not aware of the repetition. So the instead pointcut method is used. It checks the condition (which can be based on the returned result or an independent of A1) and either starts the activity once more or forwards the result of the last execution to the process. After the process object gets this result it instantiates the next activity A2.

3.2 Resources Perspective

Another workflow execution concern where flexibility is an essential requirement is the dynamic assignment of resources to activities. The WfMC differentiates four kinds of resources: human (person), organizational unit, role (e.g. the function of a person within an organization) and system (i.e. automated machine resource) [13]. As proposed in [15] the workflow resource model can be separated into the static meta model, the dynamic assignment rules and access synchronisation mechanisms. Although the frequent changes in the resource meta model are extremely seldom and, therefore, can be realized by redesigning the application, they can also be implemented by using aspectual *introductions*. It is a mechanism provided by the general purpose aspect language AspectJ [8] that allows extending given types or certain objects with additional member fields and methods.

Much more undecided and, therefore, changeable units are the dynamic assignment rules also referred to as policy resolution. They handle the resource assignment to process activities at runtime. In the Workflow Management Facility Specification of the OMG an object implementing the `WfAssignment` interface is responsible for linking `WfActivity` with `WfRessource` objects. It selects appropriate resources according to the given activity context and other process independent information. Although eligible resources are selected dynamically, the used resolution policy depends on the `WfAssignment` object the activity is related to. But often the assignment strategy itself has to be changed or extended dynamically. In this case reusable aspects can be used to either replace the assignment objects or extend the activity selection procedure by inserting additional code before or after it. Possible extensions can consider the actual workload (e.g. appropriate aspects can be dynamically added in overload situations) or history dependent assignment either in order to take advantage of personal experience or to ensure equal work partitioning [2]. On the other hand it can be necessary to replace a resource either for all assignments resp. activities (e.g. if an employee is absent and his work has to be delegated) or only for selected ones (e.g. for security reasons). This changes can also easily be realized by adding an aspect intercepting the resource invocations.

The third component of the workflow resource model mentioned above is the synchronisation of the concurrent access of multiple activities to a single resource. Since synchronisation of concurrent threads was the first application of AOP and the main purpose for the specification of the domain specific aspect language COOL [9], the suitability of aspects in this area doesn’t needs no further elucidation.

3.3 Auditing Perspective

Monitoring and logging of workflow executions as well as a comprehensive evaluation of recorded audit trails is an essential part of workflow management, since it closes the workflow development cycle comprising workflow identification,

modelling, implementation, execution and controlling [11]. The main tasks of workflow auditing are acquisition of execution data, its analysis and the utilization of the results. Both acquisition and utilization can be differentiated in short and long term, as well as active and passive approaches.

In the OMG Workflow Management Facility the acquisition of execution data is realized by the `WfEventAudit` and its subtypes which record certain types of workflow events (e.g. process or activity start and termination, context and result changes etc.). But this scheme means a passive way of acquisition, even if using the OMG Notification Service as proposed in [10], because only events published by workflow execution objects can be received. An active acquiring component can obtain arbitrary information it is interested in. It can be achieved by implementing the acquisition with the help of aspects. In the case of object-oriented (and especially OMG compliant) implementations all the relevant execution events can be detected by appropriate pointcuts. An arbitrary replacement or combination of multiple aspects without any modification of execution objects provides the necessary flexibility. For example the short term acquisition aspect providing an order processing status for a customer can be combined with an aspect implementing a long term history logging.

Using aspects modules implementing different analysis methods for audit data can be dynamically added or replaced. While passive utilization implies simple recording and/or visualisation of the results an active approach intends an intervention in the workflow execution. Long term utilization means changes to the process models that influence all future executions. Short term intervention concerns the current running process instances and can be realized by aspects adding or replacing activities and modifying control flow as described in the section 3.1, or changing the context data.

4 Summary

In this position paper we proposed an approach for the dynamic evolution of workflow instances by using aspects. It allows flexible process adaption and reuse of both the object-oriented process implementation and the adopting aspects. The changes can either be caused externally or triggered by the auditing component that can be realized by aspects too. In that way a cyclic workflow improvement can be realized.

Unfortunately the most implementations of aspect languages only support static aspect weaving at the pre-compile time (a good overview is e.g. offered by [4]). Though if using this languages a workflow has to be restarted, in order to be changed, the aspect-based adaption still allows the reuse of both primary workflow implementation and adapting aspects. Dynamic run-time aspect assignment is desirable, in order to realize automatic improvement cycle. The approaches allowing dynamic weaving are e.g. *AOP/ST* [1], that makes use of reflective capabilities of Smalltalk, or *Aspect Moderator Framework* [4], which is implemented in Java and introduces a special design pattern for objects aspects are assigned to. General purpose aspect language *Sally* [5] is an extension of Java realized by a pre-compiler. It also supports dynamic aspect assignment at the run-time.

Other potential application areas like process error handling are to be examined in the future. An open implementation issue is the aspect realization in distributed environments, which is especially important for workflow management systems.

References

1. Boellert, K.: On Weaving Aspects. In: Proceedings of the Aspect-Oriented Programming Workshop at ECOOP'99.
2. Bussler, Ch.: Policy Resolution in Workflow Management Systems. In: Digital Technical Journal, Vol. 6, No. 4, Maynard, MA: Digital Equipment Corporation, 1995.
3. Casati F.; Ceri S.; Pernici B.; Pozzi G.: Workflow Evolution. In: Proceedings of the 15 the International Conference on Conceptual Modeling, ER'96, Cottbus, Germany. Springer Verlag, Lecture Notes in Computer Science, 1996.
4. Constantinides, C.; Bader, A.; Elrad, T.: A framework to address a two-dimensional composition of concerns. In: Proceedings of the First Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems at OOSPLA'99.
5. Hanenberg, S.; Bachmendo, B.; Unland, R.: An Object Model for General-Purpose Aspect Languages. To appear in the Proceedings of the Third International Conference on Generative and Component-Based Software Engineering (GCSE) 2001.
6. Hanenberg, S.; Unland, R.: Concerning AOP and Inheritance. In: Mehner, K., Mezini, M., Pulvermüller, E., Speck, A.(Eds.): Aspect-Orientation - Workshop. Paderborn, Mai 2001, University of Paderborn, Technical Report, tr-ri-01-223,2001
7. Jablonski, S.; Bussler, Ch.: Workflow Management. Modeling Concepts, Architecture and Implementation. International Thomson Computer Press. London et. al. 1996.
8. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.: An Overview of AspectJ. To appear in ECOOP 2001.
9. Lopes, C.: D: A Language Framework for Distributed Programming. A Ph.D. Thesis. College of Computer Science. November 1997.
10. Object Management Group: Workflow Management Facility Specification, Version 1.2. April 2000.
11. Rosemann, M.: Workflow Monitoring and Controlling. In.: Jablonski, S.; Boehm, M.; Schulze, W. (Eds.): Workflow Management. Development of Applications and Systems. Heidelberg 1997, pp. 201-210. (in German).
12. Schmidt, R.; Assmann, U.: Extending Aspect-Oriented-Programming In Order To Flexibly Support Workflows. In: Lopes, C.; Murphy, G.; Kiczales, G.: Proceedings of the Aspect-Oriented Programming Workshop at ICSE'98.
13. Workflow Management Coalition: Interface 1: Process Definition Interchange Process Model. Document Number WfMC TC-1016-P. Version 1.1. October 29, 1999
14. Workflow Management Coalition: Terminology & Glossary. Document Number WFMC-TC- 1011. February 1999.
15. zur Muehlen, M.: Resource Modeling in Workflow Applications. In.: Becker, J.; R zur Muehlen, M.; Rosemann, M.(Eds.): Proceedings of the 1999 Workflow Management Conference "Workflow-based Applications" in Muenster. November 1999.