The Interrelatedness of Component-oriented Systems And Workflow-Management

What they can do for each other

R. Schmidt, U. Assmann, P. Biegler, R. Kramer, P. C. Lockemann, C. Rolker
Forschungszentrum Informatik Karlsruhe (FZI) and Universität Karlsruhe
email: rschmidt@fzi.de

The problem

Increasing the productivity of software system development and augmenting the flexibility of software systems to react to business process changes has become a dominant concern for companies competing in the global marketplace. Taking their clues from traditional production techniques, IT systems should be constructed from prefabricated, easily marketable components that can be widely reused. An example of such a product is the ubiquitous screw, which is used in a large number of products.

Software components should be context-independent in order to build and evolve component-oriented systems [CiSc96]. Context independence means that components are easily transferable from their development context to a wide variety of application contexts and they can be easily replaced by other components with similar functionalities but different or better qualities. And indeed, modern software architectures based on ActiveX/DCOM [Chap96], CORBA [OMG] or Java Beans [Beans] do support the development of systems from independently developed software units (components).

To be easily transferable from one context to another, the component must be able to interact with all potential contexts. Therefore, interaction partners should not be directly addressed by a component and the interaction protocol should be exchangeable. To solve the first issue, component-oriented systems avoid the direct interaction of components and instead use indirection mechanisms, such as the interface repository for indirection. If a component is capable of supporting multiple interfaces concurrently, flexibility is enhanced even more by allowing it to play several roles. On the other hand, the second issue, the independence of components from interaction protocols still leaves much to be desired. Presently, component-oriented systems standardize simple protocols such as method calls or event propagation, but do not provide mechanisms for using more complicated and application-specific protocols.

At first glance, one would expect object-oriented architectures to meet the needs of component-oriented architectures. After all, the properties of components and those of objects appear to be quite similar. If this were indeed so, one could employ a wealth of established methods and techniques for component-based architectures. Unfortunately, though, object orientation fails on both accounts [LoWa95], [AWBB93] . The first problem is that addressing is always direct, the calling object has to have prior knowledge of the called object and its interface. Second, the interaction protocols are not independent entities separate from the objects, but dispersed amongst all objects participating in the interaction. Therefore changes to the interaction protocols often require changes to several objects and hamper system evolution. Keeping such changes consistent is a further problem [AWBB93]. The overall control flow can only be comprised of the combined control flows of individual objects. Furthermore, such objects with an embedded part of an interaction protocol can only be reused in a new context, if the same interaction protocol is used by the other objects. An example of the inadequacies of object-oriented methods is found in the implementation of business processes. Object-oriented methods intermix single operations with the global control flow responsible for the sequence of the operations. Neither is it easy to reuse objects in other business processes, because they also contain a part of the global control flow; nor is it easy to change the global control flow, because it is embedded in a multitude of objects.

In response, several attempts have been made to fix these deficiencies. Alliances [LoWa95] separate interaction protocols from the objects. Composition filters [AWBB93] realize interaction protocols by filtering the messages sent between the objects and by providing support for error detection and synchronization. The aforementioned weaknesses of object-orientation are symptomatic of a much broader problem, the “tangling of aspects” as described in [KILL97], [Kicz96]. Embedding the interaction protocols into objects can be seen as the mixing of the interaction aspect with other aspects, such as the operational one.

A more general approach can be found in the concept of aspect-oriented programming [KILL97], [Kicz96]. It goes beyond untangling the interaction protocols from the objects and extends the idea to other aspects, such as error handling, distribution, etc., which usually are also intermixed when using traditional object-oriented methods. The concept promotes the idea of separately describing and programming the different aspects of an application and combining them via an aspect weaver.

Position

Although component-oriented system development has taken its cues from object orientation it has evolved into an independent technology. Our thesis is that we should continue taking our cues from object-oriented methods and overcome the tangling of aspects in component-based systems by transferring the concept of aspect-oriented programming to component-oriented systems.

We argue that the decomposition of the problem domain into aspects, as proposed by [Berg97] will play an important role in applying the idea of aspect-oriented programming to component-oriented systems. Aspect-oriented programming should be combined with aspect-separated domain models, such as workflow-models. Applying the separation of aspects found in workflow-management-systems will provide clues as to how constitute components which could then be utilized in a multitude of business processes. On the other hand, workflow-management systems may profit from component-oriented systems. Our goal is to fuse component-oriented systems and aspect-oriented programming paying special consideration of domain-specific aspect-oriented models such as workflow models, in order to obtain the best of both worlds.

Our rationale is that workflow-management-systems (wfms) can be seen as a kind of domain-specific aspect-oriented architecture, achieving their high flexibility in supporting business processes by separately handling the different aspects of the workflows as described by [Jabl96]. Basically, workflow-management-systems perform a strict separation of the control aspect, that means the sequence of tasks from the tasks themselves, i.e. the operational aspect. However modern approaches extend the separation to other aspect like the data aspect or the organizational aspect describing the allocation of resources to actors. On the other hand, although workflow-management-systems offer a high degree of flexibility, their enterprise-wide use is hampered by several major deficiencies. First, they rarely achieve the necessary performance and reliability to support large numbers of processes [AAAM97]. This is due to a centralized architecture, like the one described by [WfMC], creating bottlenecks and single failure points [SBMW96]. Second, the ability of wfms to work in heterogeneous and distributed environments is rather limited [GeHS95]. We are convinced that many of the limitations of workflow-management-systems can be solved using component-oriented-systems based on e.g. CORBA. A first attempt in the direction of a fully distributed workflow-management-architecture is CompFlow [ScAs98] which already follows an aspect-separated approach.

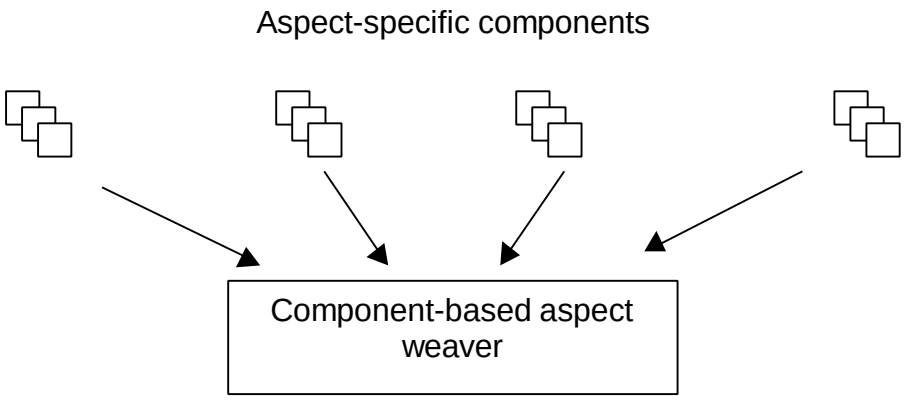

Figure 1: Component-oriented aspect weaver

In summary, application development for component-oriented systems (see also Figure 1) should start with an aspect-separated domain model such as an workflow model. Ideally components should implement only functionality belonging to one aspect of an application, and applications should be composed of aspect-separated components. By applying the concept of aspect-oriented programming there is a much better chance

that components become truly reusable, because they have to fulfill only the requirements of one aspect and not a combination of several ones. Component-oriented systems built in an aspect-separated manner can then be expected to become more flexible and evolution-transparent, because changes which concern only one aspect, only influence implementations of one aspect. To combine the components, the aspect weaver [KILL97] should be adapted for combining components. This could be done by using software cocktail mixers [AsSc97]

References

[AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan: Functionality and Limitations of Current Workflow Management Systems. To appear in IEEE Expert (Special Issue on Cooperative Information Systems), 1997.

[AsSc97] U. Aßmann, R. Schmidt: Towards a Model For Composed Extensible Components. Workshop Foundations of Component-Based Systems, Proceedings, Zurich, Switzerland September 26, 1997

[AWBB93] M. Aksit. , K. Wakita, J. Bosch , L. Bergmans and A. Yonezawa : Abstracting Object Interactions Using Composition Filters. In Object-Based Distributed Programming, R. Guerraoui, O. Nierstrasz and M. Riveill (eds.), LNCS 791, Springer Verlag 1993

[Berg97] L. Bergmans: Aspects of AOP: Scalability and application to domain modelling. TRESE project, University of Twente & STEX. http://www.parc.xerox.com/spl/projects/aop/aop-meeting-pps/bergmans.html

[Beans] Javasoft: Java Beans Specification 1.0 A. http://splash.javasoft.com/beans/-beans.100A.pdf

[Chap96] D. Chappell: Understanding ActiveX and OLE. Microsoft Press. Redmond 1996

[CiSc96] O. Ciupke, R. Schmidt: Components As Context-Independent Units of Software. WCOP 96, Linz 1996. Special Issues in Object-Oriented Programming. Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP96. Dpunkt.verlag, Verlag 1996

[GeHS95] D. Georgakopoulos, M. Hornick, A. Sheth: An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. In Distributed and Parallel Databases, Kluwer Academic Publishers, September 1995.

[HPMS97] J.Hernández, M. Papathomas, J. M. Murillo, F, Sánchez: Coordinating Concurrent Objects: How to deal with the coordination aspect ?

[Jabl95] Jablonski, S.: Workflow-Management-Systeme. International Thomson Computer Press. Bonn 1995

[JaBS97] S. Jablonski, M. Böhm, W. Schulze: Workflow-Management: Entwicklung von Systemen und Anwendungen -Facetten einer neuen Technologie. Dpunkt Verlag 1997.

[Kicz96] G. Kiczales: Aspect-oriented programming. ACM Computing Surveys, 28(4), Dec. 1996.

[KILL97] G. Kiczales, J. Irwin, J. Lamping, J.M. Loingtier, C. V. Lopes, C. Maeda, a. Mendhekar: Aspect-Oriented Programming. Position Paper from the Xerox Parc Aspect-Oriented Programming Project.

[LoWa95] P.C. Lockemann, H. D. Walter: Object-Oriented Protocol Hierarchies for Distributed Workflow Systems. In [PaTo95].

[OMG] http://www.omg.org

[PaTo95] R. Pareschi, M. Tokoro: TAPOS Theory And Practice of Object Systems. John Wiley, New York. Volume 1(1) SPECIAL ISSUE: 1995 European Conference of Object Oriented Programming

[SBMW96] W. Schulze, M. Böhm, K. Meyer-Wegener: Services of Workflow Objects and Workflow Meta-Objects in OMG-compliant Environments, OOPSLA 96, Workshop on Business Object Design and Implementation, San José, CA.

[ScAs98] R. Schmidt, U. Assmann: Compflow. ACM Symposium on Coordination languages. Atlanta 28.2.98.

[Schm97] R. Schmidt: Component-based systems, composite applications and workflow-management. Workshop Foundations of Component-Based Systems, Proceedings, Zurich, Switzerland September 26, 1997

[WfMC] Workflow Management Coalition: http://www.aiai.ed.ac.uk/project/wfmc/