Component-based systems, composite applications and workflow-management

Rainer Schmidt

FZI Karlsruhe, DBS
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
Tel: (49) 721 9654 -733
Fax: (49) 721 9654 -709
email: rschmidt@fzi.de

Abstract

Although workflow-management-systems offer a highly suitable basis for business process reengineering, they are rarely used enterprise-wide. One reason is the performance deficiencies due to limited scalability and unreliability, which are caused by an inappropriate, centralized architecture. The other reasons are the incapability of wfms to work in highly heterogeneous environments and their lack of interoperability with legacy applications. Both are caused by the underlying software architecture which improperly supports the cooperation of independently developed software units and their evolution over time. The latter problem can be solved by component-based systems. Furthermore, component-based frameworks like JavaBeans or ActiveX support the cooperation of components even in distributed and heterogeneous environments. They also provide numerous technologies for integrating legacy applications and data. However, the problem of poor performance and reliability remains unsolved. Therefore a new wfms-architecture has been developed. It uses so-called composite applications consisting of components bound together in an application document and cooperating over events. This architecture allows the distributed processing of workflows in distributed and heterogeneous environments, and thus enables the enterprise-wide use of wfms.

1 Workshop Goals

My goal during workshop participation is to show that component-based systems are the missing link between workflow-management and software engineering. Component-based systems allow the flexible development of applications by separating the different aspects of development, in the same way that wfms handle the different aspects separately, especially the control aspect. Furthermore both component-based systems and wfms follow the same approach, building applications from pre-built components. Due to critcism and the response to my work from workshop participant I will try to further develop the wfms-architecture described in section 3.4. I hope to find an impetus to increase the precision of the architecture and to learn about modeling methods suitable for component-based systems.

2 Background

Since beginning my work at FZI, a research institute at the University of Karlsruhe, I followed the scientific research in the area of component-based software engineering and participated actively in WCOP96 [CS96]. The combination of workflow-management and software-engineering is my primary area of research. Therefore I participated in several workshops of the AK-Workflow, a working group of German researchers in the area of workflow-management. Furthermore, I contributed to the book summarizing the work of the AK-Workflow [JBS97].

3 Position

3.1 The problem

Business process reengineering [Ham96] is being used by an increasing number of enterprises to increase their competitiveness. They change their structure from a tayloristic to a process-oriented one to respond faster, more flexibly and efficiently to changing customer requirements. An important means for achieving this goal are workflow-management-systems (wfms) [JB96]. They allow the definition, execution and monitoring of workflows with both human and machine-based actors. Workflows processed in wfms can be described as different aspects, representing discrete modeling domains [Jab95]. Important aspects are the functional aspect, describing the workflow's "task" and its composition of sub-workflows, the control aspect, responsible for directing the workflow execution, the data aspect, covering the data exchange between sub-workflows, the operational aspect, describing the operations performed in the sub-workflows; and the organizational aspect, representing the relation to the organizational structure of the enterprise. Because these aspects are imperative in order to execute a workflow; they are also called imperative aspects. There are also aspects of workflows, which are not necessary to execute them, but increase the quality of their execution. They are called optional aspects. One example is the historical aspect, covering the workflow execution history.

Workflow-management systems are more flexible than traditional applications in supporting process-oriented organizations, because they handle the aspects described above separately and allow for easy adaptation to changed workflows. Traditionally structured applications, however, contain only

one "hard-wired" workflow, with no possibility of changing it without modifications to the software. Wfms also differ because they are reflective systems. That means, they "know" about their own state and are capable of changing their behavior accordingly. Furthermore Wfms connect the execution of a business process with the organizational structure of the enterprise. By using role models, wfms can easily transfer work from one person to another. They assure a higher process-quality, because business rules are enforced. Better informed decisions can be made, because wfms are able to collect all the relevant information, the so-called process context, and offer it to the person who makes the decision.
Although wfms offer an impressive list of advantages, their enterprise-wide use is hampered by three major deficiencies. First, they rarely achieve the necessary performance and reliability [AAAM97] to support large numbers of processes. This is due to a centralized architecture, like the one described by [Coa], creating bottlenecks and single failure points [SBMW96]. Second, the ability of wfms to work in heterogeneous and distributed environments is rather limited [GHS95]. Their functionality cannot be transparently distributed across different systems and networks and therefore limits the usefulness of the wfms. Third, to enable enterprise-wide workflows, wfms have to cooperate with a multitude of applications, called legacy applications, in order to perform operations like bookings, etc. The crux of the problem is the software architectures used to build the wfms. They are not appropriate to support the interoperation of independently developed software units, like the wfms and the legacy applications and databases. Furthermore, these architectures do not support the evolution of software systems. However wfms and legacy applications do evolve independently from each other and therefore cause the need for permanent adaptations.

3.2 Terms

3.2.1 Components

Components are independently developed software units which can be composed without knowledge how they have been implemented. To use a component, only the knowledge of its specification is necessary, but not their implementation. This is achieved by an interface that hides the implementation and its implementation model, for example its class hierarchy. This is an important step beyond object-orientation, because the interface is decoupled from the implementation by a distinct class. Because the implementation details of a component are inaccessible for the user, there have to be mechanisms to provide information about the component's specification. The same is necessary for the adaptation of a component to individual requirements which must be possible without knowing about their implementation or changing the implementation. To achieve this goal, components offer two mechanisms, introspection and specialization. Introspection allows to get knowledge about the component's specification without knowing its implementation. Java Beans, for example, have two mechanisms which support introspection. Reflection and the BeanInfo class. Reflection makes it possible to know about all methods following a certain naming pattern without further programming. Whereas reflection is a relatively simple, automatic mechanism, the BeanInfo class gives complex information about a bean, but has to be implemented explicitly. Specialization allows the adaptation of a component without access to its implementation. Specialization in Java Beans is supported by property sheets and customizers. Property sheets offer a simple specialization mechanism by setting the parameters of the bean. Complex specializations can be done by customizers. They can be thought of as an assistant, guiding the adaptation of the bean.

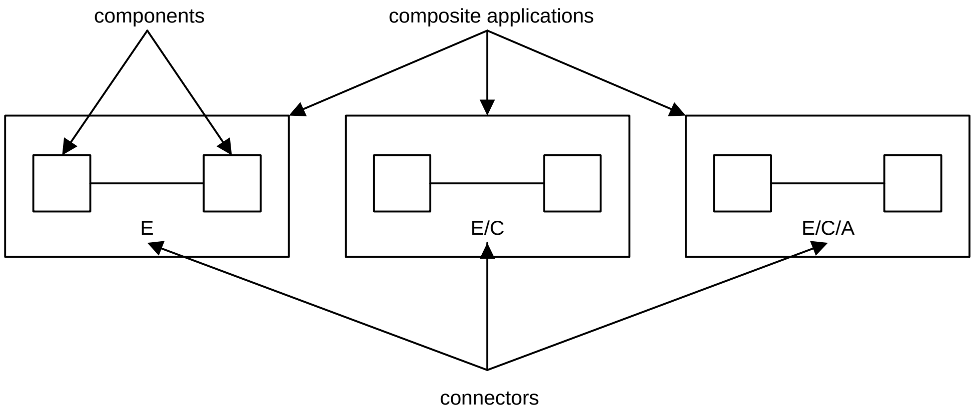

Figure 1: Components, composite applications and connectors

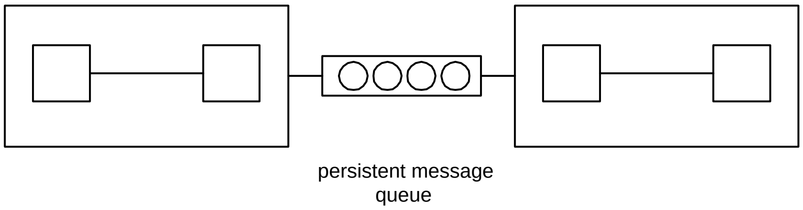

Figure 2: Event propagation using persistent message queues

3.2.2 Composite applications

Composite applications play an important role in solving the architectural problem of wfms described above. Composite applications consist of an application document, containing components which interoperate using events. They can be found in many component-based frameworks like Java Beans[Jav97] or ActiveX/DCOM [Cha96]. Composite applications can be hierarchical, which means, a composite application can be embedded in another composite application as a component. Event processing within composite applications is done via connectors, which allow the events sent to a sink to be filtered, dependent on the fulfillment of a condition for example. Only when the condition is evaluated as true, the event is propagated. Furthermore, the connector can execute an action in response to an event. At this point, Event/Actions (E/A) and Event/Condition/Action-rule s (E/C/A) can be implemented (see figure 1).

The propagation of events can also used for connecting different composite applications. Composite applications differ from approaches based on event-dependencies like the ones described in [CR90], because there are no global events. Therefore they avoid the problems of these approaches, such as combinatorial explosion of the event dependencies. The rule-processing in composite applications should not be confused with traditional rule-based systems either. Composite applications use only one rule to process the events sent from one bean to another instead of sending global events to a global-rule base. Using centralized rule-processing would create a bottleneck. Furthermore, rule-based systems have many problems like indeterminism, unpredictable behavior and performance problems.
The reliability of the event propagation can be enhanced by using persistent message queues (see figure 2), which assure event propagation even in case of a system shutdown. Furthermore, asynchronous operations allowing the integration of disconnected clients are possible.

An example of composite applications are applications built from Java Beans (see figure 3). They

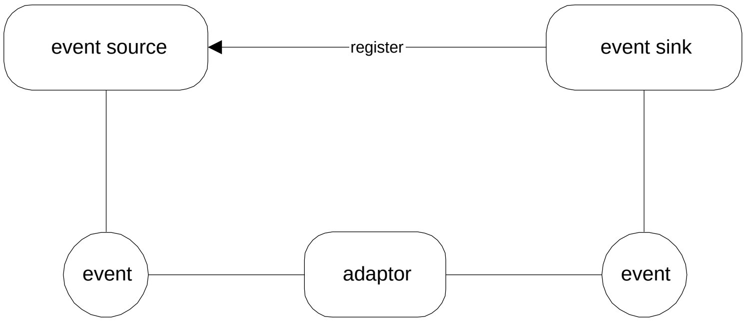

Figure 3: Event processing with Java Beans

use the event mechanism of Java. It consists of the event source, event sink and an optional connector, called the adaptor in Java Beans. The sink indicates its interest in an event by registering itself to the source or an so called adaptor. After registering, the event source can send events to the sink. If the event sink registered an adaptor, it is possible to implement E/C and E/C/A semantics as described above.

In the simplest case, the events in a composite application carry no information, except for type information. However, since events are often implemented as method calls, as in Java Beans for example, they can also carry information from one component to another.

3.2.3 Modelling composite applications

In order to model the behavior of composite applications, it is necessary to represent their hierarchical structure, therefore the composition of models must be possible. Furthermore, conditional event propagation, which is optionally augmented with the execution of actions has to be represented. State-charts [Har88] are a commonly-used formalism for describing the behavior of reactive systems. They augment traditional automata by allowing the creation of hybrid state automata and hierarchical states. This feature can be used to represent the combination of beans in the form of a composite application and the combination of simple composite applications into to more complex ones. Petri-nets [Pet81], for example, do not allow the composition of nets. Therefore, they are not appropriate for the modelling of composite applications. Furthermore, state-charts allow the definition of conditional transitions, which are necessary to represent the connectors (or adaptors in Java Beans).

3.3 Significance of the problem for component-based systems

Using components, component-based systems built from them, and component-based-frameworks like (Enterprise) Java Beans [Jav97] as an implementation basis for a wfms solves two of the problems mentioned above. The functionality of the wfms can be distributed more easily in an distributed and heterogeneous environment. Furthermore component-based frameworks improve the interoperability of wfms with legacy data and applications. They address the problem of interoperability between independently developed components like the wfms and legacy applications, which also evolve over time. However, the third obstacle for an enterprise-wide use of the present wfms, the lack of performance and reliability, can only be solved by using new, decentralized architectures.

Therefore research in component-based systems should search for a new wfms architecture, allowing a fully distributed architecture, integrating the solutions of the other two problems provided by component-based systems.

Another point which highlights the significance of this problem for component-based systems is the observation that there is an analogy between wfms and composite applications based on components like Java Beans. Both are composed of applications from pre-fabricated components. Wfms connect several legacy applications together in an workflow and composite applications put together several components and connect them with events. To further investigate this analogy wfms and composite applications are compared in detail.

An aspect-oriented view as described for wfms in the problem section is also possible for composite applications. The functional aspect of composite applications describes their semantics and the composition of individual components. There is also a control aspect which can be realized by connecting the components by events and a data aspect by passing data between components. The operational aspect can be thought of as services offered by the components of the composite application. However, there is no organizational aspect, as there is no means to represent the relation to an organization within composite applications. The analogy between workflows and composite applications is also confirmed by the fact that workflows can be modeled by State- and Activity-Charts [WWWD96].

3.4 Wfms Architecture

Using components and composite applications, a distributed architecture for supporting workflows can be developed. The functional aspect of an workflow with its hierarchy subworkflows can be represented by an hierarchy of composite applications. The control aspect of an workflow can be realized by the event processing capabilities of composite applications. The event mechanism also provides basic support for the data aspect of workflows, however more sophisticated mechanisms may become necessary if complex data structures have to be exchanged. Single beans realize the operational aspect. The organizational aspect is the only imperative aspect which cannot be realized directly with composite applications, however, serices such as the Java Naming and Directory could provide the glue to the organizational representation of the enterprise.

In this architecture the processing of workflows is done by composite applications linked over connectors and communicating via events (see also figure 4). These events also contain data or references to them, necessary for processing of the workflow. The composite applications may represent subworkflows or atomic workflows. The starting point of the processing are incoming events from other composite applications. They may have been directly propagated or may have passed a condition. They may even have caused an action as part of an E/C/A rule. The processing of conditions and actions is done by adaptors, as described above. Because the adaptors for the incoming events are outside of the composite application, they can be used by several applications, therefore reducing the overhead processing. By changing the adaptors behavior, changes to the workflow can be easily represented.

However, there may be temporal conditions, which define dependencies between the occurrence of different events. They have to be tested before the processing of the workflow can be continued, but they can only be tested inside the composite application. Therefore a checking mechanism for

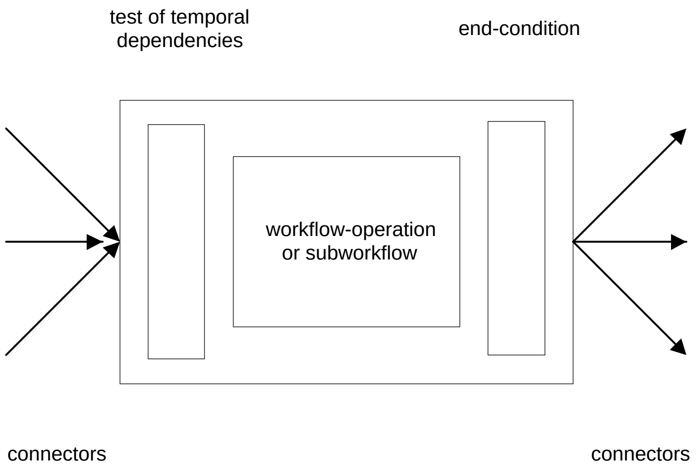

Figure 4: Workflow-processing through composite applications

temporal conditions has to be included in the composite application. After passing this test, a subworkflow or a workflow operation inside of the composite application can be processed. After this is done, the fulfillment of a condition at the end of the workflow processing is checked and events are propagated to subsequent composite applications to continue the processing of the workflow.

The architecture presented here is of a physically and logically distributed character. Because composite applications can be distributed transparently, the processing of the workflow can be physically distributed quite simply. More important, this architecture is logically distributed as well, because there are no centralized mechanisms controlling the execution of the workflow. In order to represent business processes with this architecture, it is necessary to transform business process models such as the SOM-approach [FS94] into workflow representations which are executable by the architeture described here. In order to create such a transformation, the fact that composite applications and workflows can be modelled using state-and-activity-charts will be of importance.

4 Comparison

There are several attempts to create fully distributed wfms-architectures, for example [DKM+97], [SBMW96]. However, all of them are based on traditional application-architectures, which do not use component-based frameworks like Java Beans, ActiveX and composite applications. Furthermore there are an number of attempts to use the World-Wide-Web (WWW) as basis for wfms, for example [GE96]. Those attempts use the WWW only as a platform-independent user interface, but no further functionality is provided through the WWW. However, there are first approaches for an "web-only" wfms [MPS+97].

5 Conclusion and further work

Component-based systems allow the creation of new wfms-architectures using composite applications, which enables the distributed processing of workflows in heterogeneous environments. Furthermore there are astonishing analogies between the construction of component-based systems and the creation of workflow-applications with wfms. Both cover the composition of application functionality from pre-fabricated components, whose implementation details are not know. The creation of an wfms-architecture and the transformation of business process into this architecture can be thought of as part of a domain-specific development methodology for component-based systems as proposed in [JB96].

Further work will include a more detailed description of workflow processing. Especially the event mechanism will be described in more detail and with special attention to the handling of temporal conditions and complex data structures. To create a workflow-specific development methodology the transformation of business-process models, like those of the SOM-approach [FS94] will be a major point of work.

References

[AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and limitations of current workflow management systems. *To appear in IEEE Expert (Special Issue on Cooperative Information Systems)*, 1997.

[Cha96] D. Chappell. *Understanding ActiveX and OLE.* Microsoft Press Redmond, 1996.

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

[CR90] P.K. Chrysanthis and K. Ramamritham. A framework for specifying and reasoning about transaction structure and behaviour. In *Proc. ACM SIGMOD International Conference on the Management of Data*, pages 194–203, Atlantic City, NJ, May 1990.

[CS96] O. Ciupke and R. Schmidt. Components as context-independent units of software. In *Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP96.* Dpunkt.verlag, 1996.

[DKM+97] S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah. A reliable distributed corba-based workflow enactment system for meteor2. Technical report, UGA-CS-TR-97-001, Department of Computer Science, University of Georgia, February 1997.

[FS94] O. K. Ferstl and E. J. Sinz. From business process modeling to the specification of distributedbusiness application systems - an object-oriented approach. *Wirtschaftsinformatik*, (20), 1994.

[GE96] H. Groiss and J. Eder. Integrating workflow and the world wide web. In *Workflow Management Coalition Handbook.* Wiley and Sons, 1996.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modelling to workflow automation infrastructure. *Distributed and Parallel Databases, Kluwer Academic Publishers*, 1995.

[Ham96] M. Hammer. *Beyond Reengineering.* HarperCollins Publishers, New York, 1996.

[Har88] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

[Jab95] S. Jablonski. *Workflow-Management-Systeme.* International Thomson Computer Press, Bonn, 1995.

[Jav97] Javasoft. Java beans specification 1.0 a. http://splash.javasoft.com/beans/beans.100A.pdf, 1997.

[JB96] S. Jablonski and C. Bussler. *Workflow Management.* International Thomson Computer Press, London, 1996.

[JBS97] S. Jablonski, M. Böhm, and W. Schulze. *Workflow-Management: Entwicklung von Systemen und Anwendungen -Facetten einer neuen Technologie.* Dpunkt Verlag, 1997.

[MPS+97] John Miller, Devanand Palaniswami, Amit Sheth, Krys Kochut, and Harvinder Singh. Webwork: Meteor's web-based workflow management system. Technical report, UGA-CS-TR-97-002, Department of Computer Science, University of Georgia, March 1997.

[Pet81] J. Peterson. *Petri net theory and the modelling of systems.* Englewood Cliffs (J.J.): Prentice-Hall, 1981.

[SBMW96] W. Schulze, M. Böhm, and K. Meyer-Wegener. Services of workflow objects and workflow meta-objects in omg-compliant environments. In *OOPSLA 96 - Workshop on Business Object Design and Implementation.* San Jose, CA, 1996.

[WWWD96] J. Weissenfels, D. Wodtke, G. Weikum, and A. Kotz Dittrich. The mentor architecture for enterprise-wide workflow management, 1996.

6 Biography

Rainer Schmidt is scientist at the Forschungszentrum Informatik (center for computer science research) at the university of Karlsruhe, Germany. He manages research projects in the area of workflow-management and component-based systems. He received a master in computer science at the university of Erlangen, Germany.