OOPSLA'96 Workshop on Business Object Design and Implementation II

Using Components in Workflow Activities

Stefan Schreyjak

Stefan.Schreyjak@informatik.uni-stuttgart.de

To be published in:
Jeff Sutherland; Dilip Patel (Eds.), *Proceedings of the Second and Third International Workshop on Business Object (at OOPSLA)*,
Springer Verlag, 1998

Abstract: Current workflow systems support business specialists only in the modelling and execution of business processes but not in the creation or modification of applications used in activities. By modelling applications in activities similar to workflows, specialists without or with low programming skill can create business functions of a business process and adapt these to changing market situations by themselves. Thus, the principle of self–organizing users can be realized.

Keywords: adaptability, business objects, context–sensitive applications, components, self–organization, workflow systems

1 Introduction

A *workflow management system* is a software system for the coordination and cooperative execution of work according to a specification (see also [Wor94, Jab95, JBS97]). The use of a workflow system can be divided into a planning phase, an execution phase and an analysing phase.

In the planning phase, the organizational structure of an enterprise or authority and the order of all business procedures are modelled. The whole work is splitted in single steps, which are connected though a relationship of chronological order building a process. A process step, also called *activity*, is a piece of continuous work executed by one person. It is determined for each activity which work objects (data and documents) and which human, technical, and organizational resources are necessary for the execution. The modelled business process is specified as a *workflow scheme* in a formal textual or visual workflow language. For example in [LR94, SB96a], a workflow scheme is represented as a graph whose nodes are process steps and whose edges are control and data flow relationships.

In the execution phase, the schemes are used for the controlling, supervising and recording of the activities executed. The executable instance of a workflow scheme is called a *workflow*. The *actor* of an activity can use interactive application programs to fulfil the objective of the activity. As an alternative, manual activities without computer assistance might also be possible as well as automated activities which do not need human interaction.

In the analysing phase, the recorded data is evaluated. The results extracted serve as a basis for the improvement of the workflow schemes. If all phases are passed several times in a cycle, business processes can be optimized little by little.

The use of current commercial available workflow systems is affected with several problems, because a lot of questions in the research area of workflow management are not yet answered:

Thus, it is not known whether the WfMC approach [Wor94] of a centralized workflow engine can be used to achieve the required high scalability or whether workflows should be implemented instead as business objects having built in their own lightweighted workflow engine [MSKW96]. The internet is also gaining importance for workflow systems, because it offers an infrastructure for the execution of workflows between independent organizations [LAB97]. The increasing use of mobile computer systems requires workflow systems which allow to work on activities without an on–line connection to the workflow engine [AGK95]. A lot of business processes require high fault tolerance [SB96b] which can be guaranteed only by transactional workflows [Ley95, EL97] or by advanced transaction concepts e.g. like [WR92, RSS97]. If not only strong structured workflows should be supported, there is a need for the integration of synchronous telecooperation mechanisms in workflow systems to allow the user greater flexibility [SS97]. A logical consequence is the demand for a possibility to adapt a workflow while run–time (ad hoc) to a changing situation without corrupting the security and consistence of the workflow.

All these questions and problems are interweaved with each other and cannot be solved independently. In particular, possible solutions have direct or indirect effects on the problem discussed in the next section:

The creation and use of applications in workflow activities, as well as possibilities for adaptations, like changing the configuration or behaviour, are often so restricted that requirements of a workflow system user are not met any more. This deficient flexibility can be reduced if applications are built of components and if the user gets a lot of possibilities to intervene in the programs by himself. Hence, in this paper self–organisation of the user is proposed as a basic solution principle which can be achieved using applications composed of pre–manufactured software building blocks.

In the following section problems are described in detail which result from the restriction and deficient flexibility of the applications in the activities. The principle solution method for these problems, self–organisation, is made clear by two demands which lead to a presentation of several technical solution methods. Based on those a design of a coupled system between a workflow system and a component oriented system is introduced. The paper ends with a description of existing related work and a conclusion.

2 Problem Description

In this section those problems of current available workflow systems are explained which are linked with the creation, use, and adaptation of applications in activities.

Creation of application programs

Workflow systems support the execution of (partly) automated business processes. The support is restricted to the process oriented part of a business process. The function oriented part, that are the applications realizing the business functions, are usually supposed to be given. These functions given are used via a commandline interface in the activities. When there is no application for the activity while implementing a business process, the application program must be realized in the traditional way by an order to a software vendor. The workflow system offers no support for the realization of application programs in activities.

Adaptability problem in activities

Before the emergence of workflow systems, business processes were realized with big monolithic business applications. In order to preserve marketability, business processes must be adaptable to the actual market situation. E.g. customers requests can result in the need for adapting business processes or the management wants to increase productivity by improving the process. These requests for change must be implemented by programmers of business application vendors. Hence, adaptation was costly, time–consuming, and expensive.

To solve these problems workflow systems were developed. Hereby, business processes are divided into a function oriented part — the activities — and a process oriented part — the relationship between the activities. A working task done by one actor is used as a criterion for splitting the business application into activities. These relationships are described in the workflow specification modelling the business process. Because of the more abstract representation of a workflow, the user can adapt the specification by himself and does not have to ask programmers to adapt a monolithic business application.

Workflow systems, however, only allow adaptation in the process oriented part. If the function oriented part has also to be changed, the old problem arises: adaptation in applications has to be implemented expensively by the vendor.

Heterogenous system environments

Workflow systems are used in a sometimes very heterogenous infrastructure of computer equipment, which is grown by and by. Hence, workflows have to be platform independent. The specification is already platform independent, but the applications are usually not. A workflow, e. g. for clearing of travel expenses, can be executed in principle on each platform where a portation of the workflow system exists. In reality this often fails because the applications used are not flexible enough to run on each platform. Today this problem is avoided by specifying a platform specific application for each platform in the activities. For each application used in a workflow system should exist a portation for each platform. Alternatively, applications with similar functionality can be used. Since these requirements are often not meet, activities are often only partly platform independent. If a lot of different programs exist, several problems arise, like operating errors while interacting with the application, missing functions or incompatible data formats. Altogether, this approach is unsatisfying, costly, and expensive.

Incomplete reuse of workflows
The reuse of workflows in a different enterprise requires the possibility to adapt the workflow by the user and the possibility to run the applications on all platforms used. As a consequence of the two subsections above, reuse of workflows is not really possible.

Unspecific applications
Workflow systems are especially useful when workflows are executed often. Therefore, the applications should be accommodated very well to their operation area in order to achieve high productivity. But customized applications must be bought expensively from software vendors. Instead, more generic applications, like office automation suites, are used often. But these applications can only be limited incompletely to their task domain. The user can do much more than he is allowed to do. In particular, he can make workflow data inconsistent, e. g. by deleting a document he is supposed to work on. The workflow system has only few possibilities to check what is done in activities. The incomplete adaptation to the operation area makes the task more costly and error prone than necessary. For example, the automation of tasks by specifying a mini control flow, similar to a macro, can often not be done.

3 Solution Methods

In this section a basic solution principle and some technical solution methods for the already introduced problems are described.

3.1 Solution Principle Self–Organization

The solution principle for the already introduced problems can be described by those two requirements.

1. The user should be allowed to create applications by himself.
2. The user should be allowed to make modifications in the applications by himself.

The fulfilment of those two requirements means extending the workflow modelling to application programs in activities. Applications have to be modelled analogically to workflows. From the point of view of a workflow system an activity is not a black–box any more, but it has an adaptable inner structure. Thus, activities are integrated entirely into the workflow modelling.

By accomplishing the requirements the end user gets the possibility to create, respectively to adapt to changing market situations, enterprise–wide applications supporting business processes entirely by themselves. The basic principle of self–organization by the end user within the scope of a business process can be realized in this way. The user can coordinate his working method by himself. He can adapt applications to his own requirements and he can reorganize his working method more efficient.

The whole workflow optimization cycle is enhanced with a new degree of freedom by adding self–organization. Changes in the structure of the working method can be handled more flexible and more extensive, because it is possible now to change the functioning of an application.

Self–organization supports incremental development of computer–supported business processes. In the beginning of using a workflow, it can be instantiated with functional limited applications. Single activities can be done by hand or with minimal computer assistance. The users can gain experience with his workflow and can work their knowledge into the business process by and by through changing the workflow schemes and the applications. In time, a workflow can be refined from a rough draft to a strong computer–supported, well–adapted process. This high potential for optimization allows the development of efficient business processes with assistance of the user more quickly.

3.2 Technical Solution Methods

The principle of self–organization can realized with the following technical solution methods.

Pre–manufactured software components

Instead of developing applications from scratch, pre–manufactured software building blocks (*components*) are used to compose an application without the need to know a traditional programming languages. Time–consuming development is overcome by reuse of large software units. A component hides its implementation through a public interface. Therefore, the user needs no additional knowledge in a programming language or in details of an implementation besides his knowledge as a specialist for his business. He acts with units (e. g. business objects) of this own business domain.

In a finished application built of components the components can either be identified by the end user or not seen anymore. The first kind of applications is called *component–oriented* and the second kind *component–based*. Component–oriented applications allow the end user to modify the component structure and the configuration of components.

Such a building block method is only than applicable if there exist a satisfying number of components. Components can be offered in a free market or created by a big company for their own closely specialized application domain.

Configurable components

During the development of a component it is fixed which properties of the component are configurable. These properties can modified later either by the user of the component or by the system using the component. An application built of configurable components is also configurable and therefore, adaptable to special requirements. A more efficient processing of the task of an activity is possible.

It has to be assumed that the user is supported with interactive and visual methods to get a comforting and standardized way to the configuration facilities. It should also be possible to configure an application automatically at the start. The graphical user interface, defaults for values or behaviour, respectively the amount of functions of an application, can be configurated for the special operation area of an activity.

Modelling of the control flow between components

The approach of workflow management is to extract the process management (i. e. the control and data flow) out of enterprise–wide monolithic applications. This approach is similar to the extraction of data management out of applications that has led to the development of database management systems. If applications are built of components the approach can be enhanced to component management. This is, the process management controlling the interaction of components can be extracted and modelled.

Using an explicit process model makes an application easier, quicker and more flexible changeable as if all dependencies are hardcoded in the components. The user does not have to change an application on a source code level, but can work on a more abstract design level, the model. Therefore, he does not need detailed knowledge to make changes by himself.

Such a model can be made executable. Than, the execution can be controlled and supervised by a run–time system.

Platform independent components

Platform independent components allow the execution of applications in a heterogenous computer environment. Implementing components in a platform independent language (e. g. Java, Tcl/Tk [Ous94] or Python [WvA96]) makes components and thus, applications platform independent.

Direct manipulation in a visualization

Data and documents, the user is working with, can be visualized by electronic images. Equally, the functioning of a component or the flow of a process can be visualized by a model or a view. Manipulating these images with e. g. the mouse by the user is called direct manipulation.

This principle encourages the understanding of the user for the structuring of a system and its data. The connection between the working tasks gets more transparent for the user if he is guided by a visual main thread. Hereby, he learns quicker what he is supposed to do. The main thread reduces his preparation time needing to understand what, when, and how something have to be done. Direct manipulation tries to apply familiar working methods on real entities, e. g. like files, to their electronic images.

Compound document architecture

A *compound document* [OHE96] is a container which can be filled interactively by the user with data objects of different types, like e. g. text, graphic, or database attributes. The container controls the interaction with the user and determines the layout of the objects in the document. For each data object in the document there exists a specialized component (a *data object handler*) offering certain functions on the data type, like viewing or modifying of the data. Typically, the document is stored with all included objects, respectively links to objects, in a structured data format. While data objects can be added or removed easily by the user, i. e. interactively supported by a graphical interface and without knowledge of a programming language, data object handler must be installed explicitly in the system.

4 Design of a Coupled System

In this section a design of a software system is introduced which solves the problems mentioned above by using the technical solution methods described in the last section.

The software system consists of a coupling between a workflow system and a component oriented system which is used for the creation, execution and modification of applications in activities. The workflow system takes over the process oriented part of the business process and coordinates the order of activities in a typically distributed, heterogenous computer environment. The component system takes over the non–distributed, function oriented part and coordinates working steps inside an activity. An activity gets an inner structure. It consists of a set of working steps connected with a control flow. Thus, the component system provides the functionality of a specialized, non–distributed "mini workflow system".

The component system uses a compound document architecture for the representation of business data. The user can use a compound document to integrate data belonging semantically together into one unit. The workflow system supplies the component system with the compound documents necessary for the execution of an activity. The elements of a compound document are called *business objects*. Business objects represent single entities in the business domain and have a meaning specific to business specialists. An example for a business object is the object Customer in a document Tender.

A business object consists of a pure data object, a generic data object handler and specific data object manipulators.

In *data objects* the properly data or links to it are stored. In the example the business object Customer consists of a data object which is a link to a tuple in the relation Customers stored in a relational database management system.

Data object handlers offer the user common usable (generic) functions restricted exactly to one type of a data object. Typical functions are creating, displaying, or editing of data. As a rule, data object handler can only be configured but not created by users. The creation of data object handler should be taken over by an independent software vendor. In the business object Customer a handler can be used which allows the exchange of one tuple against another tuple of the same relation.

Data object manipulators realize user specific functions on data objects. They represent the business specific meaning of a business object. Manipulators can not even be configured by the user, but also be created and modified. Manipulators are built of pre–manufactured manipulator components. Manipulator components are domain specific and can be composed by the user to application specific manipulators coordinated by a specified control flow.

The interactive or automatic creation or modification of one or more data objects through a handler or a manipulator is called a *working step*.

In the component system two kinds of control flow can be defined on two kinds of components. First, there is a control flow between methods of handler and manipulator objects, that means on working steps. Second, there is a control flow inside a manipulator between all manipulator components. In the first control flow model the user needs no programming skill, in the second model he needs skill in a scripting language.

There are two levels available for the user to modify an application in the component system. On the first level of modification no programming skill is needed. He can modify data objects using data object handlers. He can change the arrangement of data objects in a document as much as he likes, that means adding new ones or deleting old ones. He can configure data object handlers. The control flow modelled between handlers and manipulators can be changed. Available handler or manipulator methods can be added to the control flow. On the second modification level the user can modify manipulators by configuring manipulator components or by changing the control flow between these components. Though, he needs programming skill in a scripting language the control flow is modelled with.

The coupling of an application to a process is done by introducing a *process context*. Therefore, applications are supplied with a "process consciousness": They know in which context they are used and they can adapt automatically to it. The adaptation uses the configuration facility of a component and the changeable models of the component interaction. A process context is a set of activities in one or more processes associated to each other by a certain field or a purpose. The workflow system can manage several process contexts. Examples for process contexts are a set of all activities where something is signed or a set of all activities a certain group of persons (e.g. a department) is working on. Process contexts can be used for setting up a context specific default for data entry fields or for automating the choice of language the graphical user interface should appear. An application can act in a process according to its use. The realization of *context sensitive applications* is possible.

5 Related Work

Componentware [Ude94] is a term established mainly in the software industry and can be divided in three trends: compound documents, component–based development, and communication infrastructure for distributed objects (middleware). Several aspects of these trends were used and enhanced in the design.

Compound document architectures can be used not only for the creation of multimedia documents, but also for the creation of applications by using a component facility for controlling the component with a script. Each component offers its functions in a public interface and reacts on events created by the user, e.g. with the mouse pointer. Special control flow components could automate the control flow between the other components by executing scripts. Current available products (OLE or rather ActiveX [Bro95] and OpenDoc [OMG95]) do not support this field. Beside the scripting facility there is no support built in for modelling and controlling the data and control flow. The products are entirely tailored for the document metaphor where interaction between data objects is not needed. The components allow no user intervention for configurating or modifying components. Components must be built by developers with good knowledge of the compound document framework. Hence, user specific components are costly to create by the user.

The whole product family around Visual Basic components [Mic91], as well as the newer Java Beans [Sun97], concentrate on the development of applications using components, which can be manipulated in a visual programming environment. Objective is the rapid creation of applications or prototypes. The application developed gives no hint that it is built of components. Hence, the application looses its properties resulting in being built of components. The user has no more possibilities to interfere, like changing, adding, or configurating components. The application is only component–based not component–oriented. The creator of an application needs in most cases programming skill in order to realize the coordination of components, i. e. the reaction of components to events occurring.

In middleware the term of a component is much wider interpreted. There, a component is a communicating object of a programming language. Accordingly, a lot of services supporting the communication infrastructure and the development of object oriented distributed applications are delivered with typical middleware. The most important products are CORBA [Cor97] from the Object Management Group and DCOM [Rog97] from Microsoft. Such products can be used as a basic technology in the design already introduced, beyond it, component orientation for the user cannot be found here either.

Leymann proposes in [Ley95] not to call an application in an activity via the command line interface, but to call exactly one (not interactive) method of a business object. Here, a business object can be seen as a component. This approach was realized in the workflow system Surro [SB96a]. If the whole business process is taken apart in many small components in order to increase the reuse degree, the workflow engine has much to do. For example, not every method call must be assigned to different actors, therefore the engine must process a lot of unnecessary activities. The refuge to realize a business process with few but big and complex components is nearly equivalent to call a whole application. As a consequence of this experience the design does not try to integrate the workflow system and the component oriented system but proposes a division into highly coupled systems.

6 Future Work

Some aspects in the design introduced are not already considered. Some questions still remain open and some future work has to be done.

It has to be investigated further for appropriate visual languages to describe the two models for the specification of the control flow between the working steps and between the manipulator components. So far, different models are used in the design for the modelling of workflows and for the modelling of component oriented applications. In order to keep the learning effort low for the user, the models should be united into a more homogeneous model.

Up to now, the design is focused on the traditional operating field of workflow systems, the indoor (or intra–organizational) use. Exploiting internet technologies, e. g. for the transport of platform independent components, would enhance the operating field for workflow systems to workflows spread above independent organisations.

Proprietary component models, like ActiveX and JavaBeans, do have or are gaining more importance. Considering the large marketplace of available components it must be thought about an integration of these components into the coupled system.

After completing the design a prototypical coupled system should be developed. This work can be based on the workflow system Surro.

7 Conclusion

Typical users of workflow systems are specialists for their field and therefore, their programming skill is usually low. Such a user can only modify the workflow specification in current available workflow systems, whereas, the applications in the activities are regarded as given and cannot be created or changed by the user. Moreover, the given applications are usually not equipped for the heterogenous system environment of workflow systems and often they cannot be adapted to the special purpose of an activity.

Building applications of configurable, platform independent components solves these problems. By modelling the relationship between components similar to the modelling in workflow systems even a business specialist can create and modify applications. He can change the structure of an application built of components and he can configure these components without programming skill. By introducing a process context managed by the workflow system component–oriented applications can be configured context sensitive by default, i. e. very specific to a purpose.

The possibility to adapt workflows and applications allows the users to realize the principle of self–organization in an enterprise or authority. Reuse of workflow schemes in another organization is only applicable when adaptability is possible and platform independent components are used.

These solution principles can be realized by the coupling of a workflow system and a component oriented system.

The data to be worked on are organized as business objects in a compound document. The user can add, delete or edit business objects without having programming skill. A business object consists of a pure data object, a generic data object handler and specific data object manipulators.

Data object handlers offer commonly useful functions on data object and data object manipulators offer specialist functions. While handler can only be configured, additionally, manipulators can be modified by the user visually supported if he possesses a certain programming skill. Therefore, manipulators are built of configurable manipulator components whose control flow relationships are modelled and described with scripts.

A control flow is modelled between working steps, that is a method call of a handler or manipulator object. This control flow can be modified by the user. Therefore, the inner structure of an activity consists of modelled dependencies between working steps.

The user has several possibilities to intervene in the software behaviour which supports him to fulfil the objectives of an activity. He can add new data and functionality and up to a certain degree, he can modify existing functionality.

References

[AGK 95] Alonso, G. ; Günthör, R. ; Kamath, M. ; Agrawal, D. ; El Abbadi, A. ; Mohan, C.: Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System. **In:** *Proc. of 3rd Int. Conference on Cooperative Information Systems*. Wien, 1995

[Bro95] Brockschmidt, Kraig: *Inside OLE 2*. Microsoft Press, 1995

[Cor97] Object Management Group (OMG): *The Common Object Request Broker Architecture and CORBA 2.0/IIOP Specification (Revision: 2.0)*. Februar 1997. – http://www.omg.org/corba/corbaiiop.htm

[EL97] Eder, Johann ; Liebhart, Walter: Workflow Transactions. **In:** Lawrence, P. (Hrsg.): *Handbook of the Workflow Management Coalition*. Wiley & Sons, 1997, S. 195–202

[Jab95] Jablonski, Stefan: Workflow-Management-Systeme: Motivation, Modellierung, Architektur. **In:** *Informatik Spektrum* 18 (1995), S. 13–24

[JBS97] Jablonski, Stefan ; Böhm, Markus ; Schulze, Wolfgang: *Workflow-Management: Entwicklung von Anwendungen und Systemen*. dpunkt Verlag, 1997

[LAB97] Lavana, H. ; A., Khetawat ; Brglez, F.: Internet-based Workflows: A Paradigm for Dynamically Reconfigurable Desktop Environments. **In:** *International Conference on Supporting Group Work '97*. Phoenix, Arizona, USA, November16–19 1997

[Ley95] Leymann, Frank: Workflows Make Objects Really Useful. **In:** *Proceedings of the 6th International Workshop on High Performance Transaction Systems (HPTS)*, 1995. – http://www3.hursley.ibm.com/hpts95/proc95.htm

[LR94] Leymann, Frank ; Roller, Dieter: Business Process Management With FlowMark. **In:** *Proc. 39th IEEE Computer Society Int. Conference (CompCon)*. San Francisco, Cal. : IEEE, Februar 1994, S. 230–234

[Mic91] Microsoft Corporation: *Microsoft Visual Basic*. Redmond, Washington : Microsoft Press, 1991

[MSKW96] Miller, J. A. ; Sheth, A. P. ; Kochut, K. J. ; Wang, X.: CORBA–Based Run–Time Architecture for Workflow Management Systems. **In:** *Journal of Database Management* 7 (1996). – Special Issue on Multidatabase, to appear

[OHE96] Orfali, Robert ; Harkey, Dan ; Edwards, Jeri: *The essential distributed objects survival guide*. New York, NY [u.a.] : Wiley, 1996

[OMG95] OMG: Compound Presentation and Compound Interchange Facilities / OMG. 1995. – Forschungsbericht Doc. No.95-12-34

[Ous94] Ousterhout, John K.: *Tcl and Tk Toolkit*. Massachusetts : Addison Wesley, 1994

[Rog97] Rogerson, Dale: *Inside COM*. Microsoft Press, 1997

[RSS97] Reuter, Andreas ; Schneider, Kerstin ; Schwenkreis, Friedemann: ConTracts Revisited. **In:** Jajodia, S. (Hrsg.) ; Kerschberg, L. (Hrsg.): *Advanced Transaction Models and Architectures*. Kluwer Verlag, 1997

[SB96a] Schreyjak, Stefan ; Bildstein, Hubert: Beschreibung des prototypisch implementieren Workflowsystems Surro / Universität Stuttgart, Software–Labor. Fakultätsbericht Nr. 1996/19

[SB96b] Schreyjak, Stefan ; Bildstein, Hubert: Fehlerbehandlung in Workflow–Management–Systemen / Universität Stuttgart, Software–Labor. Fakultätsbericht Nr. 1996/17

[SS97] Schneider, G. ; Schweitzer, J.: Closing the gap between synchronous and asynchronous cooperative work. **In:** *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97)*, 1997

[Sun97] Sun Microsystems: Java Beans / JavaSoft. 1997. – Forschungsbericht Version 1.01, http://java.sun.com/beans/spec.html

[Ude94] Udell, Jon: Componentware. **In:** *Byte* 19 (1994), Nr. 5, S. 46–56

[Wor94] Workflow Management Coalition: The Workflow Referenz Model / Workflow Management Coalition. 1994. – Forschungsbericht Document Number WFMC-TC00-1003, Issue 1.1, http://www.aiai.ed.ac.uk/project/wfmc/DOCS/refmodel/rmv1-16.html

[WR92] Wächter, Helmut ; Reuter, Andreas: The ConTract Model. **In:** Elmagarmid, A. K. (Hrsg.): *Database Transaction Models for Advanced Applications*. San Mateo : Morgan Kaufmann, 1992, Kapitel 7, S. 219–263

[WvA96] Watters, Aaron ; van Rossum, Guido ; Ahlstrom, James: *Internet Programming with Python*. MIS Press/Henry Holt publishers, 1996