Second International Workshop
on Component–Oriented Programming

Coupling of Workflow and Component–Oriented Systems

Stefan Schreyjak

Stefan.Schreyjak@informatik.uni-stuttgart.de

Published in:
Wolfgang Weck; Jan Bosch; Clemens Szyperski (Eds.), *Proceedings of the Second International Workshop on Component–Oriented Programming (WCOP '97)*,
TUCS General Publication No. 5, September 1997,
ISBN 952-12-0039-1, ISSN 1239-1905,
`http://www.tucs.abo.fi/publications/general/G5.html`

Abstract

The use of component–oriented systems in a workflow system solves several problems of today's workflow systems like e. g. lacking adaptability. For that purpose, however, both systems must be modified and coupled in a two level system. In this paper several modifications are proposed that will improve the systems' cooperation, like the creation of interfaces and the provision of a process context, as well as the introduction of interactively changeable views in the application.

1 Introduction

Workflow systems and component–oriented systems have similar assignments and use similar technical approaches to facilitate the development of applications. But they differ in the application domain and in the magnitude of the application. Workflow systems control enterprise–wide applications that consists of activities. Component–oriented systems control local applications that consists of software building blocks. This paper is going to show how both systems can benefit from each other.

First, basic concepts and terms of workflow management are introduced. Some definitions in the area of component–oriented programming follow. In section 3 several problems are described with which today's workflow systems are confronted. These problems motivate the use of component–oriented programming in workflow systems. A pure combination of both systems does not result directly in the desired success. The newly emerging inadequatenesses are explained in detail in section 6. In the following section a coupling of the systems instead of a combination is demanded. This is proved by proposing an approach for each itemized problem how the problem can be solved by a modified workflow system and a modified component–oriented system. In the last section all essential ideas are summed up.

2 Workflow Management Systems

This section introduces essential concepts and terms [WfM96] of workflow management, as well as terms of component–oriented programming.

A *workflow management system* (WFMS) is a software system for the coordination and cooperative execution of business processes in distributed heterogeneous computer environments. The objectives of a WFMS are in a first phase the modelling of the structure of an enterprise and of the sequence of all business procedures, and in a second phase the controlling, supervising and recording of the execution of all modelled processes.

A sequence of business procedures is modelled in a *business process*. All sequential or parallel relationships between process steps are specified in business process models. It is determined for each step which work objects (data and documents) and which human, technical and organizational resources are necessary for the execution. A business process is modelled as a graph with process steps as nodes and control and data flow relationships as edges. An executable process model is called *workflow*.

A process step, also called *activity*, is a piece of continuous work executed by one person. The *actor* of an activity can use an interactive application program 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.

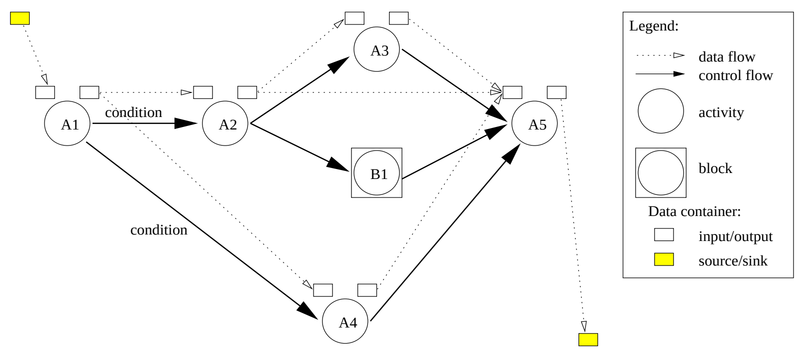

Figure 1: Model of a business process

Figure 1 shows a model of a business process which is used in IBM's workflow system "FlowMark" [LR94] and in Surro [SB96a]. All activities are connected by control flow connectors. Each activity has an input and an output container used for input and output data. The transport of data is controlled by data flow connectors.

Figure 2 shows the inner structure of an activity. The input data container consists of a set of variables which are set in the beginning by data flow connectors. The application can read data from the input container and write data into the output container. Incoming control flow connectors are evaluated in the start condition. If this condition is true, the activity waits for the occurrence of an event. When the event has occurred or no event

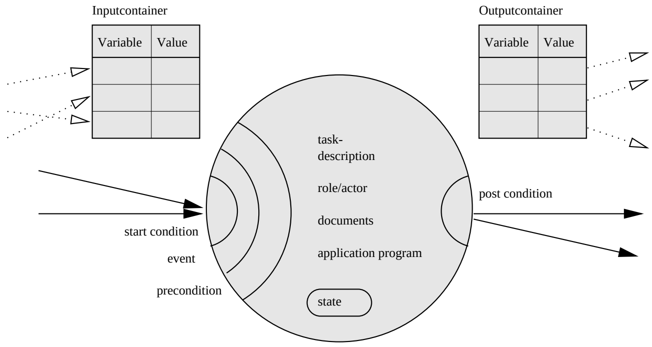

Figure 2: The inner structure of an activity

was specified, the precondition is tested. If it is true, the activity is assigned to an actor and an entry is put on his work list. The actor can then start the program associated with the activity. The application processes the data in the input container and writes output data. The workflow system can test with the postcondition if the task of the activity has been accomplished successfully. If this is not the case, the activity is return to the same actor.

The central component of a workflow system is the workflow engine. It reads the workflow specification, instantiates the workflow, dispatches the scheduled activities to the actors and transports data and documents to them. Application data and documents, as well as workflow management data, are stored in a database or document management system. The engine uses an organization module, which knows the whole organizational structure of the enterprise, to cast a role, e. g. that of a clerk to a real person. This module can also be used as a generalized resource planner. The specification of the business process and the organizational structure are created with a visual workflow editor. A workflow information system shows several views on the actual state of the workflow system and allows to view even the history of the processing. An activity manager is used as an actor's user interface to the workflow system. The actor uses this program to start workflows and activities allocated to him.

Workflow systems are mainly used in enterprises with highly structured business processes which are executed very often. Users of a workflow system are actors incorporated in the process and work managers who model the business process. Work managers must have programming skills. They remain, however, specialists for the business process, not for the programming.

Programs which are assembled using components can be executed in activities. Because this domain is of great importance some terms of *component–oriented programming* (COP) are described in the following to clarify how the terms are used in this paper:

Components are reusable software building blocks which can be assembled or disassembled in an application without violating its integrity. There are atomic and composed components. Atomic components offer functionality of one specific domain, i. e. functionality must be distributed among components in a way that interfaces are of minimal size.

A *component model* specifies an architecture which enables components to interact and consists of a set of protocols. Components are instances of one component model.

A *component–oriented system* is an architecture for a software system which provides the infrastructure for components. An implementation of this architecture consists of an application development environment and a runtime environment for components. A *component–based system* uses components only in the development environment. In the runtime environment (respectively the compiled application) components do not exist anymore.

The term *component–oriented application* defines an application which is built using components as building blocks and is used to distinguish itself from applications conventionally built.

3 Problem Description

The use of current workflow systems reveals some problems. In this section all those problems are discussed which can be solved or reduced using component–oriented programming.

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, adaption is costly, time–consuming and expensive.

These facts have resulted in the development of workflow systems. Hereby, business processes are divided into a function–oriented part — the work steps — and a process–oriented part — the relationship between the steps. One step in the work of an 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 the workflow, the user can adapt the workflow more easily than programmers can adapt their monolithic business application.

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

Heterogenous System Environments

Workflow systems are used in a very heterogenous infrastructure of computer equipments, which is grown by and by. Hence, workflows have to be platform independent. The specification is already platform independent, but the applications are not. Therefore, an activity needs one platform specific application for each platform. Every application used in a workflow system needs to exist in several portations: one application for each platform. Alternatively, applications with similar functionality have to exist for each platform. 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 of adapting the workflow by the user and the possibility to run the applications on all used platforms. 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, generic standard applications are often used. But these applications can only be limited incompletely to their task domain. The user is allowed to do much more than he should. The workflow system has only few possibilities to check what is done in activities. The incomplete adaption to the task domain makes the task more costly and error prone than necessary. For example, automation of mini control flows can usually not be done in this generic applications.

No Mutual Profit of Services

Standard applications have no knowledge about their use in a workflow system. So they cannot use services offered by the workflow system. Vice versa, the workflow system sees activities as black boxes. It is not able to control the applications in the activities. As a consequence of this lacking arrangement, services may be implemented twice.

4 Combining Workflow and Component Systems

An approach is to use component–oriented programming for the development of applications in activities. Workflow systems are a large and rewarding application domain because the complexity of program logic in activities is typically low and often similar. It turns out quickly that today's systems are not prepared satisfactorily for that use if they are not modified for that purpose. So, the combination of these unmodified systems shows the problems mentioned above in a different perspective:

- A simple combination does not eliminate the problem of lacking mutual profit.
- Now, the user has the possibility to create customized applications with components providing standard functions. Unfortunately for every new adaption (like a different color) a new additional application is needed. In addition, applications cannot be customized dependent on the task domain (e. g. a certain activity in a process).
- The problem of heterogenous system environments is not solved if the components remain platform dependent.
- Reuse will be possible when the problems of heterogeneity and adaptation will be solved.
- The applications in the activities can be adapted by the user if component–oriented systems are used. This is not valid for a component–based system! Indeed the possibility to adapt applications is not supported very well from today's component systems. The user cannot adapt an application easily:

◇ Functionality dealing with the cooperation of components can only be gathered with difficulty. Events and callbacks are used as a communication technique. The code of these callbacks is spread over all components. Therefore, the user must accumulate the knowledge of the structure of a component–application in a reengineering phase by himself. There is no support by the system. Moreover, often only one structuring element exists, the hierarchy of GUI–widgets (controls).

◇ The replacement and addition of components is not supported in an adequate way by today's component systems. There are no integrity checks whether all required interfaces and preconditions are met when using a new component. The user has to test it by himself.

◇ The glue code is necessary for the creation of callbacks and allows the cooperation of components. Glue code is usually written in the same language with which the component is implemented, and not in a language which is tailored to user skill. Such a language should be easy–to–learn, easy–to–understand and easy–to–use. Most of all the code must be interactively testable.

5 Coupling Workflow and Component–Oriented Systems

The combined use of unmodified systems solves the problems mentioned only partly. Therefore, it has to be examined if it would not be more successful not only to combine but also to couple both systems. In such a coupling both systems can be modified and enhanced with additional functionality to solve these problems. Objective of the coupling is to allow transfer and adjustment of similar technologies for a redundance–free partitioning of responsibilities and to allow a highly effective cooperation. The coupling results in a two level system where each single system can be used on its own, too.

In the following subsections, a solution is proposed for each problem mentioned. Also, some technical aspects concerning implementation aspects are discussed.

To: **No Mutual Profit of Services**

Both systems must be modified in a way that they offer their services in public interfaces. Double implementations can be avoided if it is defined which system implements the service and which system uses the service. A replacement of one system is only possible when these interfaces are standardized.

An example for such an API are controlling functions for components. A controlled abort, a resume, as well as storing and loading of component states are important functions to support error handling of the workflow system [SB96b].

An example for the distribution of functionality is the transport of software. With the help of internet protocols a workflow system can transport and install applications at the actor's computer [SB96a]. This functionality can also be used for the transport of components, just like the loading of java applets in a html–page.

A more advanced example for distribution is the realization of persistent applications. Workflows are persistent because activities are considered atomar units. Component–oriented applications are generally not persistent, but in the application domain of workflow management persistent applications are desirable because an actor may want to change his working place within a running activity. For that purpose complete persistence is not necessary, all what is needed are some persistent states. Hence, a component–oriented application can be divided into activities such, that the workflow system is able to support the persistence. The component system does not need to implement it, too.

To: **Unspecific Applications**

Component–oriented applications need to be aware of the surrounding process to be able to customize to a specific task. This can be achieved with a process context.

A *process context* is a logical storage used by components to store and retrieve data. The workflow system provides and manages different contexts. Contexts can be defined through conditions. For example, a person can form a context, but also a workflow instance or an activity in a workflow template. The concept of a process context allows *context sensitive* applications.

A component can store the attributes' values, configuration scripts or other data in the context. For example, a context sensitive command history can be implemented. Default values in data entries are another example. Thus, an actor can use the old data he had typed in last time using this application.

A context database can be compared with the X–resource database, with the restriction that only one context exists which is identified via the application name or class. The data container concept can also be seen as a primitive process context.

To: **Heterogenous System Environments**

The problem of heterogeneity can be overcome if the component system and the components are implemented in a platform independent programming language. The object–oriented language Java was designed for that purpose. But a number of other languages exist which also possess that property, like Tcl/Tk [Ous94] or Python [WvA96].

Proprietary component models like ActiveX and OpenDoc, as well as platform dependent components can be used in a component system through proxy components or bridges. A proxy delegates all accesses to the proper component which runs in a component system on a remote node. This should be transparent to any user of the system.

To: **Adaptability Problem in Activities**

Analogous to the modelling of workflows, application must be modelled to allow easy–to–use adaption for the users.

The user can change an application built with components on three adaption levels: The first level is limited to changes of the component attributes. On the second level glue code used for the component communication can be adapted to change the cooperative behaviour of the component–application. These two levels can be controlled by the process context. On a third level the user can replace and add whole components and change the structure of the component–application.

To perform this adaption work the user needs easy–to–use editor tools. Editors for component attributes can be found in many existing component systems. Editors for the glue code consist of conventional programming editors. No editor exist for the communication structure. Here, graphical methods are necessary to describe the functionality of the whole application on a level of high abstraction. Tools for a structural change can be found in interface builders common to all component systems, but the structure of the application is identical to the hierarchy of the GUI–elements.

Adaptations made by the user have to be stored. The scope of adaptations may differ. Global adaptations are valid for all users, while local adaptations are only visible in a certain context.

The understanding of a component–application's functionality can be improved by the introduction of views. A *view* shows a certain aspect of the application. The user can use a view to change something interactively in the component–application. The following views has been identified up to now:

- The GUI–view shows layout and structure of the GUI–hierarchy.
- The structure view shows the hierarchy of components in form of a tree.
- The model view visualizes the functionality of (composed) components. A model of a component abstracts from implementation details and facilitates the understanding for the user. Now, the user can make adaptations at the design level instead of the implementation level by giving access to the glue code.

  An open question remains as to which visualization models are suitable for the user group. The research domain visual programming and object–oriented design can give a stimulus to that. Enhanced state charts [Har88] would be one possibility currently being considered.

6 Related Work

Leymann proposes in [Ley95] a method to build business applications in an object–oriented manner through removing all flow dependencies from business objects. Flow control will be managed by a dedicated control object. Adler [Adl95] calls this coordination model process planner. He had implemented control objects with the help of tcl scripts. Leymann uses a workflow system as a control object and calls this method "heavyweight scripting" in contrast to "lightweight scripting" found in typical script languages like tcl, perl, rexx, etc. The workflow system Surro [SB96a] realizes this concept. The workflow engine uses an ORB to call business object methods in an activity. Therefore, Surro can be seen as a component–oriented system with business objects as components. These concept *integrates* a workflow system and a component system into a single level system. For each activity a corresponding component exists and the engine is used as a central control unit.

The flexibility of such business applications benefits from process modelling. But it results also in bad scalability. Using small components increases the number of components and consequently the expenditure of the centralized workflow engine. The engine has to handle each component as an activity. Therefore, only a small number of complex components are used as business objects which model typically real entities of the business domain, like application forms or business accounts.

Consequently, the integration concept seems not to be adequate for small components. System requirements of workflow and component systems are too different to be combined in a single level system.

7 Conclusion

In today's workflow systems the adaptability of workflows ends at platform dependent activities. This results in bad reusability. Furthermore, control over applications is minimal and applications cannot use workflow services. Therefore, potential for automatism is not used.

A first approach to improvement is to use component–oriented systems in activities. But a simple combination of unmodified systems does not succeed satisfactorily. There is a need for higher coupling.

The creation of a two level system–coupling requires several modifications in both systems. The following modifications have been proposed in this paper. The introduction of interfaces will allow redundance–free partitioning of system tasks. The introduction of a process context will enable the use of highly specific applications (context sensitive applications). By using interactive editable views, modelling the functionality of an application built of components, the user can more easily adapt these applications.

Integration of both systems into a single level system results in bad scalability. In the next project steps the approaches mentioned will be enhanced and evaluated in a prototype.

References

[Adl95] ADLER, Richard M.: Distributed Coordination Models for Client/Server Computing. **In:** *IEEE Computer* 28 (1995), April, Nr. 4, S. 14–22

[Har88] HAREL, David: On Visual Formalisms. **In:** *Communication of the ACM* 31 (1988), May, Nr. 5, S. 514–529

[Ley95] LEYMANN, Frank: Workflows Make Objects Really Useful. **In:** *Proceedings of the 6th International Workshop on High Performance Transaction Systems (HPTS)*, 1995

[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

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

[SB96a] SCHREYJAK, Stefan ; BILDSTEIN, Hubert: Beschreibung des prototypisch implementieren Workflowsystems Surro / Universität Stuttgart, Software–Labor. 1996. – Forschungsbericht Fakultätsbericht Nr. 1996/19, Software–Labor Bericht SL–5/96

[SB96b] SCHREYJAK, Stefan ; BILDSTEIN, Hubert: Fehlerbehandlung in Workflow–Management–Systemen / Universität Stuttgart, Software–Labor. 1996. – Forschungsbericht Fakultätsbericht Nr. 1996/17, Software–Labor Bericht SL–3/96

[WfM96] WFMC: Workflow Management Coalition Specification — Terminology and Glossary. http://www.aiai.ed.ac.uk/WfMC/DOCS/glossary/glossary.html. Juni 1996. – Document Number WFMC-TC-1011, Issue 2.0

[WvA96] WATTERS, Aaron ; VAN ROSSUM, Guido ; AHLSTROM, James: *Internet Programming with Python*. MIS Press/Henry Holt publishers, October 1996