# Workflows Make Objects Really Useful

**Frank Leymann**
**IBM Software Solutions Division**
**German Software Development Lab (GSDL)**
**Hanns-Klemm-Str. 45**
**D-71034 Böblingen**
**Germany**

e-mail: frank_ley at vnet.ibm.com

***Abstract:*** *One possible perception of workflow management systems is that of a means to realize advanced, distributed application systems. An application developed according to the workflow paradigm consists of a process model and a collection of program granules. As a result, the application becomes flow independent, and its encompassed program granules become subject of reuse. Since object technology strives towards reusability too, the relation between both technologies is often asked for: We show how workflow technology can be beneficially used from the very beginning during object oriented analysis and design, how transaction handling is eased, and how workflow management systems can even enhance the reusability of objects.*

**Keywords:** Business objects, business processes, object technology, reuse, transaction models, workflow technology.

## 1 Introduction

The WfMC (Workflow Management Coalition) is striving towards standardized interfaces to enable both, interoperability of workflow management systems, and to allow programs to be written is such a way that they can be used within different workflow management systems [26]. The expectation is that the latter will stimulate vendors to provide software components being reusable across the variety of implementations of workflow management systems. This in turn will allow to compose applications out of these components via models of business processes.

Recent efforts of the OMG within its BOMSIG (Business Object Management Special Interest Group) and its vertical industry SIGs show the increasing importance of business objects as granules of reuse. A joint cooperation between OMG and WfMC is targeted towards a standard allowing the exploitation of business objects in workflows. The result will facilitate a straightforward composition of applications out of business objects.

This meeting of the workflow paradigm and the object paradigm is very likely just the initial step towards a mutual large-scale exploitation of both technologies, especially within the area of component based software construction: Together, the advent of industry standards for models of business processes and those ready to use "off-the-shelf" components will accelerate the exploitation of workflow technology. The graphical features for editing process models typically provided by workflow management systems [10] will allow for an easy customization of the resulting applications. Inherent features of the object paradigm like polymorphic business objects will make customization even more flexible.

We show in this contribution that workflow technology is a powerful vehicle to enable the widespread exploitation and reuse of objects (or components, respectively): For this purpose we first remind the fundamental benefit of workflow technology, i.e. the enablement of flow-independence and its consequences on application structures is emphasized. Next, we discuss the dual role of a WFMS in an object request broker environment namely as an application object exploiting object services as activity implementations, and as an object service itself providing a framework which allows the exploitation of objects in business processes. Various techniques used in object oriented analysis and design to depict dynamic behavior are abstracted and it is outlined how the modelling features of WFMSs can be beneficially used in these phases. We outline the different roles a WFMS can play in terms of X/Open's reference model of distributed transaction processing and the benefits originating from the provision of the resulting transactional services. Finally, we sketch a transaction model which provides compensation based partial backward recovery in WFMSs to support "business transactions", and the benefits of business transactions on objects.

## 2 Removing Flow Dependency

Nowadays, it is general knowledge within the information processing community that the insight into the so-called *data dependency* of applications and its negative consequences [4] was the primary impetus for the wide-spread exploitation of database management systems that can be realized today. Applications are much more flexible and less vulnerable against changes at the data organization level when leaving data handling with a database management system and exploiting its accompanying features.



In: Proc. 6th Intl. Workshop on High Performance Transaction Systems (HPTS) (Asilomar, CA, September 17 - 20, 1995)

### 2.1 The Notion of Flow Dependency

Today, a similar observation can be made: It becomes more and more clear that contemporary applications are obstructions to change the business processes in which enterprises serve their customers. Large application systems contain the code that directly represents these business processes themselves. Consequently, changing a business process (e.g. to become more competitive) requires changes of the affected application systems.

While the algorithmic knowledge of how to perform a particular step of a business process reflects a proper application functions, it is the knowledge of sequencing these steps (i.e. the flow of control) and distributing them to the responsible organizational units (i.e. the flow through the organization) together with the appropriate information (i.e. the flow of data) what reflects the proper business processes ([13], [14]). It is exactly the latter which makes the application vulnerable with respect to changes of the underlying business processes: Applications are *flow dependent*.

### 2.2 Workflow Based Applications

The exploitation of workflow technology allows to separate algorithmic knowledge from knowledge about business processes. The knowledge about the potential sequencing of algorithmic parts and the technical details of their execution, distribution, and data exchange (the so-called *process model*) is left with the workflow management system, allowing to extract the corresponding code from an application, consequently removing its flow dependency. What results is called a *workflow based application* consisting of a process model and a set of programs. These programs are implementations of the proper algorithmic components of the underlying business process and directly support its basic execution steps (the so-called *activities*).

Within a workflow based application, required changes of the business processes are easily achieved by adapting the associated process model. Most frequently, no changes of the encompassed activity implementations (i.e. the basic execution steps) are necessary, revealing that no code has to be modified to reflect the change of business processes. The above mentioned lack of flexibility is removed, i.e. a workflow based application is flow independent and thus more robust.

As a remark only we mention that one can expect beneficial impacts on the productivity of a corporation's application development staff from building workflow based applications: Once the process model is defined, the interfaces between the programs implementing the activities of the business process are defined. In principle, implicit assumptions about data routing, program sequencing and states are removed. Thus, parallel development becomes more effective.



### 2.3 Flexible Implementation Structure

The workflow based application paradigm immediately allows for a very flexible implementation structure: Since the programs may be dispersed on different machines, running in heterogeneous environments it becomes a *networked* application. Moreover, the binding of each activity to its supporting program is dynamic, i.e. it occurs at run time based on the definitions of the workflow [15]; in this sense the locations where the various pieces of the applications are executed are transparent and can be changed at run time. This is another aspect of increased flexibility of applications exploiting workflow technology ("topology independence"). Because the workflow manager does not assume any inherent structure of a program associated with an activity such a program may even be implemented as a client/server structure; in this sense, the client/server paradigm and the networked application paradigm are independent from each other.

## 3 Object Technology

Enterprises are investing today in object technology to improve the productivity of their programmers and to enable even non-data-processing professionals to build applications. Application building will become more and more component based. We describe now the granules that can be used (i.e. ideally *re*-used) for application construction, and how and when workflows can be beneficially exploited.

### 3.1 Granules of Reuse

Based on object technology the most common granules of reuse are class libraries, frameworks, parts, and design patterns. From a reuse perspective they differ as follows (e.g. [8], [22]):

- Class libraries provide for reuse at the finest level exploiting especially subclassing, overriding, and overloading. The emphasize of reuse within class libraries is on implementation, i.e. on code. Reusing class libraries typically requires coding within a programming language.
- Frameworks allow for reuse at a coarser level by ignoring many details of the underlying class library, externalizing only the aspects relevant to the subject application area. Often, classes of the frameworks are used as implemented, sometimes they will be subclassed, or frameworks encompassing abstract classes have to be completed. What is reused within frameworks is a mixture of design and implementation.
- Parts are used exactly as they are provided, especially, they are already completely implemented. The reuse of parts emphasizes a component based application construction. Typically, this construction is performed via a *scripting* language, i.e. a "usage language" (in contrast to the "implementation language" of the parts themselves).

- Design patterns emphasize the reuse of design. A design pattern provides the documentation of a solution for a small architectural problem and explains the intend, trade-offs, and consequences of the chosen design. Since its documentation is independent from a particular programming language, a design pattern must be implemented each time it is used. Thus, it cannot be embodied in code and is only of help for programmers.

As described in (2.2), activity implementations for process models are typically flow independent and free of assumptions about their usage (with the consequence that an external mechanism to establish transaction boundaries is required; see (4.1)): Thus, one activity implementation can be used in many different process models. If both, the activity implementation and the workflow manager comply to the WfMC standard for "invoked applications" the activity implementation can be used in many different WFMSs. As a result, the exploitation of workflow technology also stimulates reuse of code, having activity implementations (i.e. proper application logic) as granules of code reuse. While the reuse of class libraries, frameworks, parts, and design patterns is coupled with object technology reuse based on workflow technology is independent from it.

Beside the fact that activity implementations become subjects of reuse, the strong demand for reusing process models themselves origins from users of workflow technology. For this reason, many WFMSs allow for activity implementations which are again process models (*subprocesses*), thus enabling top/down and bottom/up modelling of processes especially stimulating the reuse of process models as subprocesses. The meaning of this kind of reuse will be amplified by defining industry standards for models of business processes typically performed in particular application domains; these process models can especially be reused as subprocesses in enterprise specific processes.

Also, the application of object technology can be very beneficial in defining reusable process models. For example, subtyping and polymorphism allow to define abstract process models (e.g. a "disbursement" process) by defining activities as method-invocations on abstract classes (e.g. a WITHDRAW method of an ACCOUNT class). The abstract process models are transformed into "concrete" process models (e.g. disbursement for savings accounts or current accounts, respectively) by subclassing the ACCOUNT class with a SAVINGS ACCOUNT class and a CURRENT ACCOUNT class with (obviously) different implementations of the WITHDRAW method.

### *3.2 Removing Flow Dependencies From Objects*

Some of the components built for application construction based on object technology will be *business objects*, i.e. objects which are immediate subjects of business processes (e.g. person, account, contract, business card, etc.). Especially, object technology will be exploited to build such components. Now, one of the underpinnings of object technology is the insight that robustnesss of a system is normally achieved by encapsulating things which might become subject to changes. So, if the order in which operations are to be performed can change, or if operations can be added or removed, for example, the guidelines of object technology consequently recommend a dedicated control object encapsulating operation scheduling. Thus, to achieve robustness via encapsulation not only behavior and data must be regarded (this is what is usually done) but also "ordering".

If the last proposition is ignored, then following the encapsulation paradigm tends to hide fragments of the proper business processes in the implementations of the components. This means that not only the components themselves become flow dependent, but transitively each component based application as a whole too. In addition, the business processes (having an important value by themselves [14], [24]) are only partially explicitly described and externalized to a broader community. As a consequence, implementing components in a way such that they become flow independent will result in much more flexible component based applications.

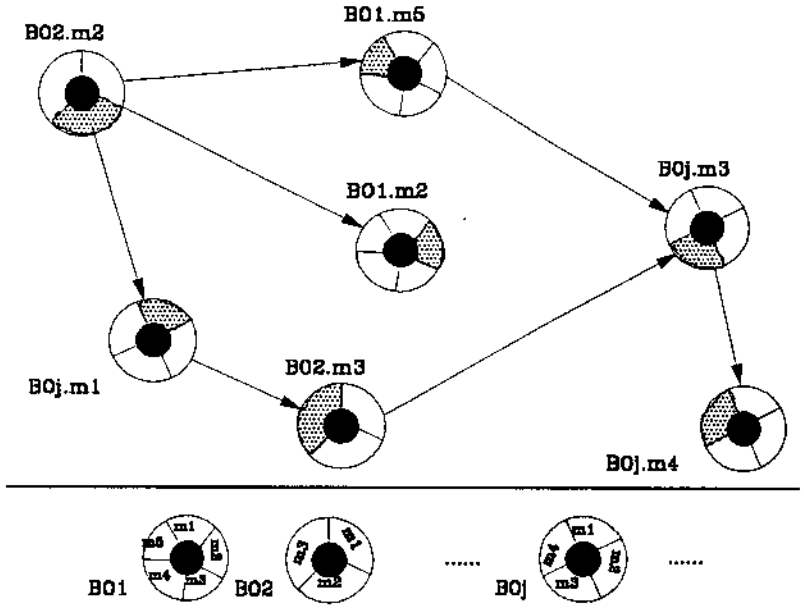


Figure 1. *Workflow As Conductor Of Objects*

### *3.3 Scripting*

Building flow independent business objects will enforce a clear separation of the

more stable behavior of the business objects from the more dynamic behavior of the business processes. A business process explicitly describes the rules of how, when and by whom the services provided by the various business objects are exploited. An activity within a business process may directly correspond to a method of a business object (shaded areas in figure 1).

When the statics of a business (i.e. its business objects in our current scenario) is split from its dynamics (i.e. its process model) the interaction between business objects is defined by the process model. The process model may be perceived as a *script* prescribing the employment of business objects to reach particular business goals. At run time the workflow management system will manage the flow of control and data between the business objects (figure 1). Even more, it will enforce that the proper organizational units of the enterprise will become responsible for utilizing the services provided by the various business objects.

In order to allow for a direct exploitation of business objects as activity implementations within business processes the workflow management system has to support the invocation of methods of the business objects. When the execution of an activity is requested the workflow management system will directly invoke the respective method. Note, that restricted to control flow C++ follows a similar philosophy: Programs written in C++ in general consist of objects and procedural elements explicitly describing the control flow between method invocations of various objects.

Because of the BOMSIG efforts of the OMG mentioned in the introduction it can be expected that many business objects will become standardized and defined via IDL (Interface Definition Language [20]). These IDL defined business objects can be implemented by using SOM (System Object Model) technology [9]. Then, by natively supporting the invocation of methods of SOM objects the workflow management system facilitates the exploitation of services provided by the corresponding business objects as activity implementations. By further exploiting DSOM technology (Distributed SOM [9]), which is IBM's CORBA [20] implementation) even the location of these objects is irrelevant to the workflow manager and objects accessible via any CORBA compliant implementation can be used as activity implementations. In CORBA terminology the workflow management system (as a scripting feature) would be an application object exploiting object services or common facilities provided by business objects (figure 2).

Also, OpenDoc as well as OLE allow the composition of applications out of components, more precisely out of parts (see 3.1). Especially, the joint presentation of the interfaces of such applications as an integrated, composed document is supported. By adding the mechanism of invoking OpenDoc parts to the workflow manager it will be possible to exploit such parts as implementations of activities too. Vice versa, OpenDoc parts representing particular workflow constructs may be provided. For example, a part handler for activities may allow for including work requests in documents and to start the execution of work directly from the document. It is the former that reveals that process models may also be perceived as scripting language (in the sense of 3.1) for using parts.

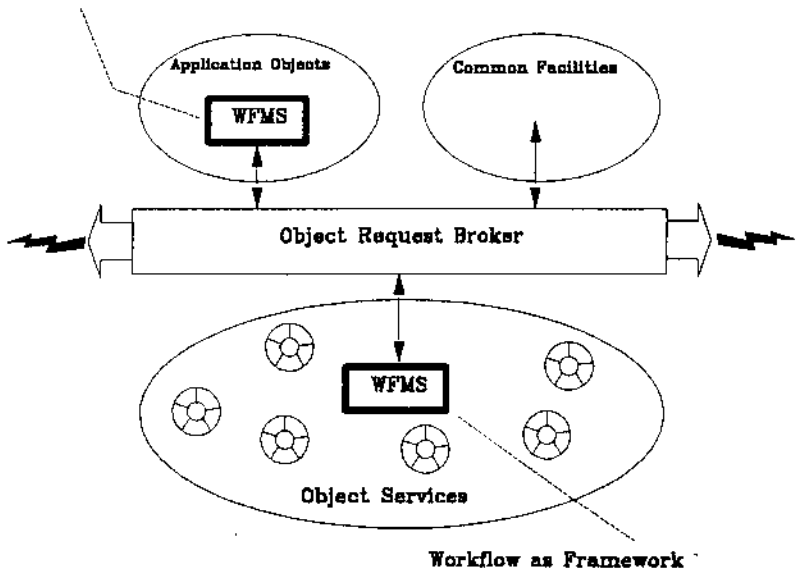


Figure 2. *Workflow Management System And Object Request Broker*

Note, that we do not argue that workflows are a substitute for scripting languages like REXX or VisualBasic. Instead, we would like to introduce the notion of "lightweight scripting" and "heavyweight scripting": *Lightweight scripting* is performed whenever a desktop application has to be composed for a single enduser perhaps by the enduser himself; a "traditional" scripting language is used for this. *Heavyweight scripting* is performed whenever an application has to be composed which requires the collaboration of multiple people distributed in an enterprise; a process model executed by a workflow management system is used for this. Heavyweight scripting adds features like parallelism, heterogeneity, distribution, context dependent (i.e. dynamic) resource allocation (e.g. staff resolution) to the notion of scripting. The implementation overhead inherent to these features is the reason why we call this kind of scripting "heavyweight" and why two different categories of scripting have their own right of existence.

### 3.4 Object Oriented Analysis and Design

Object technology provide many methodologies to capture an application domain during the analysis and/or design phase. The derived object models represent the static representation of an application domain. In order to allow for an analysis of the dynamic behavior of an application various techniques are proposed: Collaboration graphs [25], event flows [21], timing diagrams [1], interaction diagrams [11] all show at the abstract level a structure as depicted in figure 3 which can be called most appropriately *message flow diagram*. Basically, such a diagram may be perceived as a description of the control flow between method invocations. Some authors even associate additional conditions with the "arrows" describing the "control flow" (e.g. [11]) revealing the workflow aspects in the sense of [13] of such message flow diagrams.

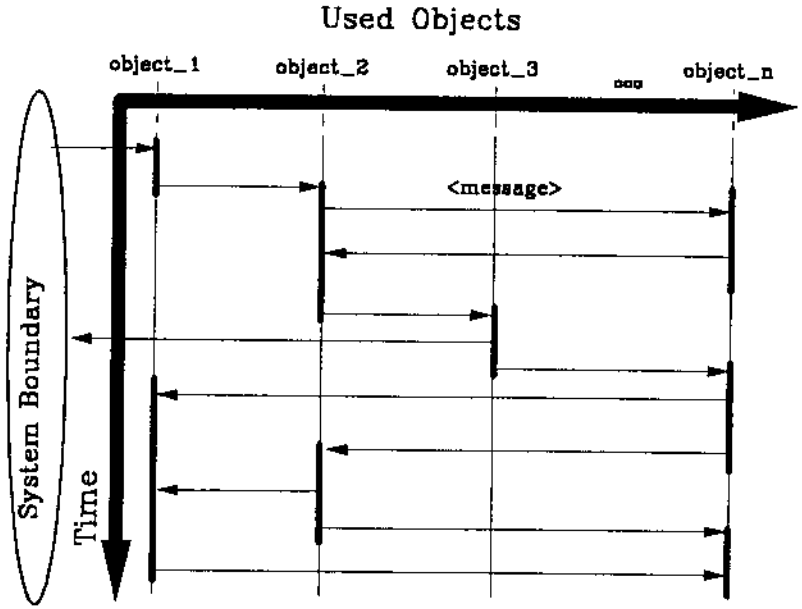


Figure 3. *Message Flow Diagram*

In [12] *use cases* represent a business from an external point of view, i.e. by "actors" representing the environment and who want to use the business. Consequently, structures not seen by actors are excluded from the use case but the interactions of the environment with the business is represented. A use case describes the flow of events necessary to yield a certain result. Also, a coarse grained object model (as a result of the analysis phase) accompanies the use case introducing the objects (of one of the category "interface", "control" and "entity" [11]) necessary to run the business. At a more detailed level (i.e. during the design phase) *interaction diagrams* show how a use case is realized by communicating objects [12]. An interaction diagram shows the stimuli that are transmitted between the objects, as well as parameters that follow the stimuli. Here, the corresponding message flow diagram contains even data flow aspects emphasizing its workflow relation [14].

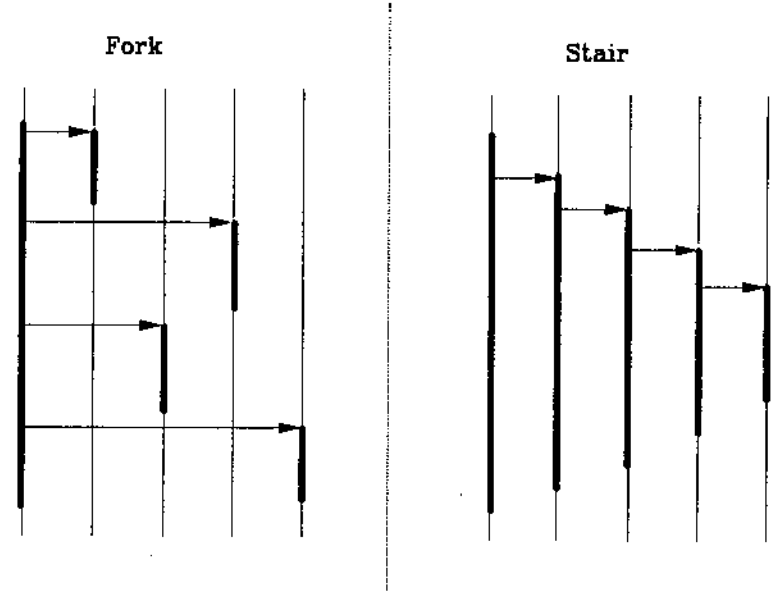


Figure 4. *Fork And Stair Diagrams*

Based on the fact that message flow diagrams represent workflow aspects a fundamental observation has been made in [11] on the structure of such diagrams: A message flow diagram is in general composed of two basically different structures namely either a *fork* structure or a *stair* structure (see figure 4).

- Fork structures are implied by centralizing responsibilities, i.e. when the global control and data flow is placed in one object. The remaining objects are used for enquiries, utilities, interfacing users etc.. Using such a kind of "control object" is an approach workflow purists will prefer.
- Stair structures are implied by delegating responsibilities, i.e. when each object only knows a few other objects and how to exploit them. In such a kind of decentralized structure the responsibility for the local control and data flow is with

each object itself; note especially that these objects are then flow dependent. The delegation of responsibility is the approach object purists will prefer.

The proposition derived in 3.2 (namely that robustness achieved via encapsulation must not only regard behavior and data but also ordering) is made manifest in forks: A fork typically encapsulates ordering. In contrast to this, stairs express assumed stability of ordering (or collections of strongly connected operations [11]). In general, fork and stair structures will be used in combination to yield a stable and robust structure.

It is one of the strengths of workflow technology to facilitate in an easy manner especially modifications of orders of operations. Thus, it is only natural to exploit workflow technology for the implementation of fork structures, i.e. for encapsulating ordering of operations: Simply, the controlling object itself becomes an instance of a process model which in turn describes the control flow between the affected objects.

In [23] a methodology has been sketched that uses process models as representations of control objects of use cases; additionally, a protocol is outlined that tights such kinds of control objects together with the (appropriately behaving) other kinds of objects (i.e. interface objects and entity objects) to result in a workflow application. Since that approach requires the exploitation of workflow technology after the analysis phase we sense obstructions from object purists against this. Because of this, our approach goes one step further: We allow modelers to stay in the object paradigm also during the design phase and we support them to map their design to workflow based applications afterwards.



The enabling factor for this is our fundamental observation that (subdiagrams of) message flow diagrams can be mapped to templates of process models (in terms of [13]). Before applying the corresponding transformation we analyze a message flow diagram and subdivide it into a collection of subdiagrams each of which is either a fork structure or a stair structure. Fork structures typically contain control objects which govern the behavior of the affected objects and are thus represented as process models in a straightforward manner. Stair structures characterize stable delegation of responsibilities and are natural candidates for modularization (as programs or as subprocesses).

The following sketches our proceeding in transforming a message flow diagram into the control flow aspects of a process model. Since our intend is to show the basic philosophy behind our transformation of message flow diagrams into templates of process models we omit to show how data flows can be derived from message flow diagrams.

1. *Identify all control objects*:

   Forks have a single control object. A control object is defined as one sending many messages out and receiving only "a few" messages back, thus the difference of outgoing messages and incoming messages must be "high". The real crux is to tune this difference appropriately to identify control objects.

2. *Build all fork structures*:

   All objects which receive a message from a single control object are included in the associated fork. Objects receiving during their life-time messages from more than one object need a special treatment (not covered in this sketch).

3. *Build all stair structures*

   Each connected component of the diagram which results when all outgoing messages of all control objects are removed is considered to be a stair structure. Especially, a control object receiving a message out of a stair is (virtually) added to the stair.

4. *Transform forks into process models*:

   The target of a message sent by a control object becomes an activity the associated implementation of which corresponds to the invocation of the method specified in the subject message of the target object. If the target is a fork structure its associated diagram is transformed into a subprocess. If the target is a stair structure and it has been transformed into a process model this process model becomes a subprocess; otherwise, the stair is considered to be a single activity with exactly the module as implementation which has been associated with the stair.

5. *Refinement*:

   Message flow diagrams in general hide parallelism. Thus, the resulting control flow structures should be reworked to enhance the degree of parallelism.

6. *Transform stairs*:

   A stair can be transformed into a process model too; from that source code can be generated (see below and [18], [19]). Otherwise, the diagram is transformed into a program with whatever is available.

In 3.3 we made the observation that C++ programs may be perceived in general as specifications of the control flow between method invocations of servicing objects. This in turn shows the desirability of a *workflow compiler* or source code generator which will transform a process model having method invocations as activity implementations into C++ programs (or DSOM programs, respectively: see [18], [19]). This is especially useful if the subject process model shows a particular behavioral pattern like for example being consecutively executed by a single person.

### 3.5 Coexistence

Object technology and workflow technology are two "orthogonal" technologies: One can exploit the other in various ways resulting in synergies valuable for both paradigms, but they can also coexist in the sense that each technology has its own area of applicability without the other.

Coexistence can simply mean that the workflow manager executes programs written in an object oriented programming language. The workflow manager's mechanism of starting programs bound with activities supports natively various invocation mechanisms, for example starting programs provided as EXE files, CMD files, or entry points in DLLs, CICS or IMS transactions, message queue applications, and so on. Since the way a program is constructed is irrelevant to these invocation mechanisms the workflow manager can start the corresponding programs regardless whether they are written in an object oriented programming language or not.

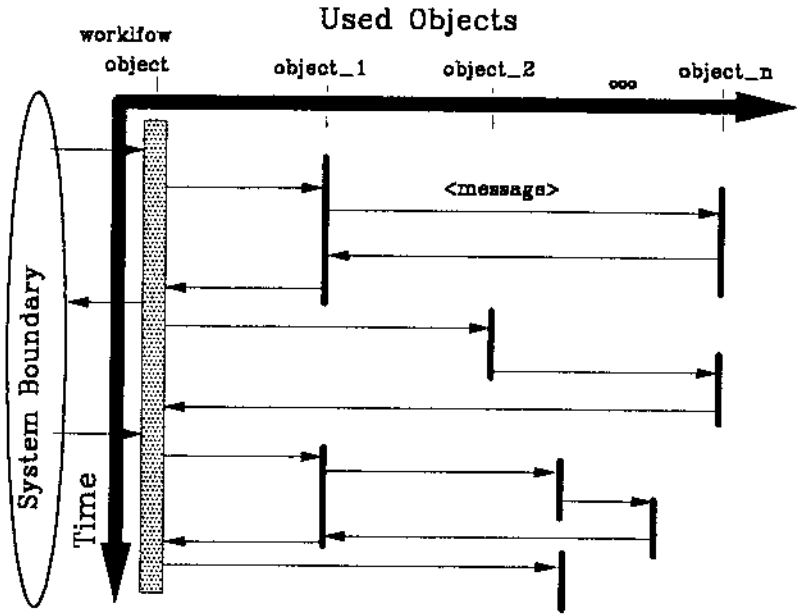


**Figure 5.** *Workflow Framework*

Another form of coexistence results when treating the workflow management system as a collection of object services accessible via an object request broker (*workflow framework*, see figure 2 and figure 5). This will allow to let application objects to participate in workflows, especially those who will not adapt the workflow paradigm.



For example, an application object which has been included into a process model may use an ACTIVITY object of the workflow framework to communicate a particular state change which might be control flow relevant for the encompassing business process. Such kind of a workflow framework is the expected outcome of the current activities performed in the cooperation between OMG and WfMC.

Business objects may be implemented in such a way that they are not externalizing state information which is relevant for determining the flow within a business process. Such a business object may wish to communicate in an object oriented manner with the workflow manager in order to influence the navigation through the business process the business object is tight in. A workflow framework providing the corresponding services as objects will allow for that. Again, exploiting SOM technology for this may be beneficial because of resulting in language neutrality and binary compatibility.

## 4 Transaction Support

With the increase of exploitation of workflow technology for application construction workflow management systems need to provide transactional services. Both, the capability to bind activities into atomic processing units is required (e.g. [6], [17]), as well as the support of a transaction model for real-world business transactions [16]. This strategically positions workflow management systems from an application point of view as the next generation of transaction management systems.

### 4.1 Atomicity

When components (e.g. business objects) are build for reuse it is a known proposition from software engineering that their coupling should be weak, i.e. the number and complexity of interconnections between components is to be minimized [5]. As a consequence, when dealing with recoverable resources each single component must be allowed to manage transaction boundaries as appropriate from its own local perspective.

For example, a business object ACCOUNT has a WITHDRAW and a DEPOSIT method. Since a customer may sometimes wish to pay in money to his account, or sometimes wish to withdraw money from his account, both, the DEPOSIT as well as the WITHDRAW method will establish its own transaction boundary. Now, the transfer of money from one account to another will reuse both, the DEPOSIT method and the WITHDRAW method by invoking WITHDRAW on the first account and DEPOSIT on the second. This reveals the necessity of a separate transaction boundary surrounding both method invocations [7]: In case the WITHDRAW or the DEPOSIT method aborts as a subtransaction the whole transfer transaction (as a superior transaction of the former transactions) must be aborted.

Thus, reusing components with separate transaction boundaries does require some flavor of a nested or multi-level transaction model. This will allow for writing components without any assumption on their use in other transactions. It has been shown in [3] how this can be achieve by exploiting OMG's Object Transaction framework [2] which of course nicely matches with business objects. For the integration of applications in workflow management systems [6] proposes enhancements of X/Open's DTP [27] and the introduction of dependency rules to WFMSs which will allow for that.

In [17] a more conservative approach is taken: It is suggested to exploit the existing X/Open DTP protocol and use dependencies implied by a simple and pragmatic modelling construct called "atomic sphere" to combine components into a unit which provides for atomicity of the executions of the collected components.

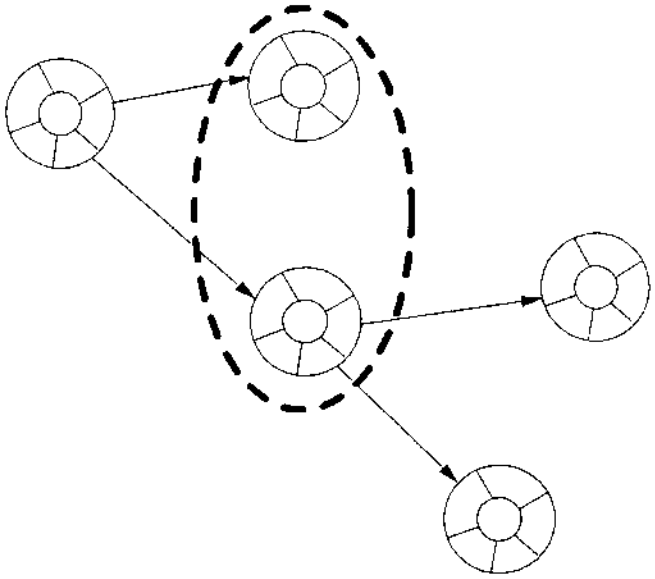

**Figure 6.** *Atomic Sphere*

At the syntactic level an *atomic sphere* is a set of activities of a process model where each activity itself ensures ACIDicity. Moreover, consider the subgraph induced by these "transactional activities" when representing a process model as a directed graph defining the control flow between activities: All paths between two nodes of this subgraph must already be paths within the subgraph, i.e. it is enforced that the control cannot leave the atomic sphere and enter it again (e.g. the activities collected via the dashed line in figure 6). The semantics of an atomic sphere requests that either all activities that have run within the atomic sphere committed or all aborted; note that due to different heuristic decisions of two participants this semantics cannot be enforced. At the operational level it is required that each implementation of an activity within an atomic sphere exploits only resource managers in the sense of [27].

As a consequence, the workflow management system appears in different roles within the structure defined in the X/Open DTP reference model:

- As a resource manager participating in the termination protocol of the coordinating transaction manager: The WFMS registers itself as a resource manager whenever it detects that the control flow enters an atomic sphere. This registration is performed dynamically (i.e. via *ax_reg()* with the TMREGISTER flag set in its **xa_switch_t** structure) in environments with a low number of atomic spheres; in other environment the registration is performed statically (i.e. via configuration file). As a result of this registration the WFMS will be asked by the transaction manager to join any global transaction started from that time on. When a global transaction is joined the participation of the WFMS within the two-phase-commit protocol run by the transaction manager will basically allow the WFMS to get information about the outcome of each activity within the atomic sphere.
- As an application controlling the root transaction which represents the atomic sphere: When the WFMS detects that the control flow enters an atomic sphere the first time it starts a transaction which will be the superior transaction for all activities within the atomic sphere. Thus, each activity implementation that runs within this atomic sphere will become a subordinated transaction to the root transaction. The WFMS will ensure that the control flow will not leave the atomic sphere before all other activity implementations within the atomic sphere have terminated or it is known that they will not become startable. In this case the WFMS will end the root transaction appropriately and the usual processing continues.
- As an application exploiting participating resource managers: The WFMS stores persistent information of the execution status of each instance of every process model in a database. Each control flow relevant state change of an activity implementation and the associated recording of that fact in the WFMS's database system must be performed in an atomic unit. For this purpose, the transaction of the WFMS recording such a state change in its database runs as a subordinated transaction to the activity implementation's transaction (which thus is treated as a superior transaction).

### *4.2 Business Transactions*

Due to the nature of business processes activities are in general long running (especially tolerating system shutdowns), must be thus interruptible, and often externalize

intermediate results. Obviously, the same is true for business processes themselves. Furthermore, a business process contains in general collections of activities which are semantically coupled in the sense that either all coupled activities must be performed successfully or the work associated with the activities must be backed out to allow the business process to continue correctly. In this context the usual transaction models (in general realized via mechanisms like locking etc.) do obviously not apply.

A transaction model that supports such "business transactions" is proposed in [16]: Since the above mentioned semantical coupling of activities is in general not expressed via control flow dependencies or data dependencies we allow arbitrary sets of activities to be coupled into so-called *spheres of joint compensation* (or *compensation spheres* for short). In case a compensation sphere has to be aborted compensation activities are scheduled by the WFMS automatically in an order which is reverse to the order in which the proper activities within the compensation sphere have been executed. For this purpose, activities must be specified as pairs consisting of the implementation of the proper activity and the implementation of the compensation activity.

Many different parameters effect the actual behavior when backing out a compensation sphere (refer to [16] for all the details): For example, it can be specified whether after compensation the work within affected process branch(es) should continue at the entry points of the compensation spheres to be backed out, or whether some administrative actions have to take place, or whether the control flow simply has to continue at the entry points of the compensation spheres without performing any compensation. Furthermore, compensation spheres can become a target of cascading backouts, and backout cannot only be performed "discrete" by running the compensation activities associated with the proper activities but also in an "integral" manner by simply running a compensation activity (which can again be defined as a process model) which is directly associated with the affected compensation sphere itself.

Interesting features result when atomic spheres and compensation spheres are combined. For example, an atomic sphere can also be defined to be a compensation sphere too. When it is detected (e.g. by specifying a 'true' **report_heuristics** parameter of the *commit*-operation of the TERMINATOR interface [2]) that a heuristic decision (which differ from the consensus outcome) has been taken by a participant represented by an activity within the atomic sphere the abortion of the associated compensation sphere can be initiated. Thus, appropriate actions can be immediately and automatically scheduled, for example, the checking for integrity violations, or the cleansing of inconsistent data.

### *4.3 WFMS-Based Transaction Management*

A workflow manager can be perceived as a means to couple applications together even if they are developed without having the other applications in mind. The common context often needed for such a coupling may be provided by the dataflow facility of the workflow manager especially providing for persistence of the required context information. What results is an application system providing added value to its user when compared with its encompassed discrete applications. For example, the correct sequencing of discrete applications in order to achieve a business goal paying regard of each particular business situation is automatically provided, idiosyncrasies of the various invocation mechanisms are hidden, information is automatically passed from one application to the other etc.

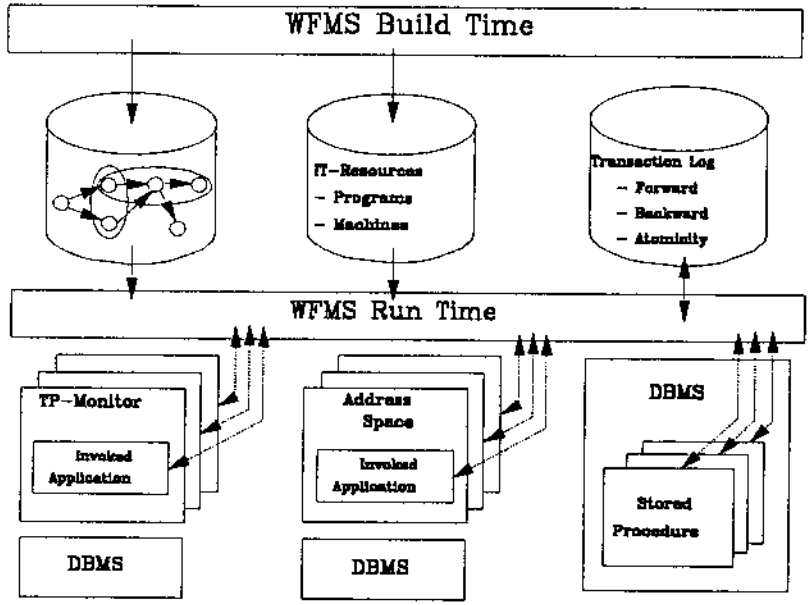


**Figure 7.** *WFMS-Based Transaction Management*

In addition to the context and state of such an integrated application itself the history of the executed discrete applications, their order, and their local context can be tracked and made persistent. In case erroneous situations occur which require to compensate already finished applications the system can -based on this persistent data- automatically schedule formerly defined "compensation steps" to correct the situation. What results is a feature that supports partial backward recovery introducing a compensation based semantic transaction paradigm for applications integrated via a workflow manager (see 4.2).

Also, a workflow manager can provide a common transactional context for applications with transactional behavior, and can manage and control the atomic outcome of collections of such transactional applications (see 4.1). By this, a workflow manager facilitates a certain flavor of multi-level transactions.

Figure 7 depicts that the coupled applications may run under the control of various transaction monitors, as native operating system programs, or as stored procedures, they may be transactions or non-recoverable units, they may run locally or distributed in heterogeneous environments: The workflow management system ties them together in the sense sketched above revealing the middleware aspects of workflow management. With atomic spheres and compensation spheres workflow management systems can be perceived as the base for future transaction management.

## 5 Summary

We have shown that workflow technology is a powerful vehicle to enable the widespread exploitation and reuse of components: The inherent potentials of workflow technology like the enablement of flow-independence, forward and backward recoverability of workflows, transactional workflows, etc. in conjunction with its embedding in object request broker environments and its exploitation in object oriented analysis and design allow for both, a seamless use of prefabricated components, as well as a seamless build for reuse of components.

**Acknowledgements:** I am grateful to Dieter Roller and Jürgen Uhl for discussing with me many of the subjects covered in this paper.



## 6 References

[1] G. Booch, *Object oriented design with applications* (Benjamin/Cummings, 1991).
[2] E.E. Cobb et. al., *Object transaction service*, OMG Document TC 94.8.4 (1994).
[3] E.E. Cobb, *The impact of object technology on commercial transaction processing*, Technical Report, IBM Santa Teresa Lab (1995); submitted for publication.
[4] R. Elmasri, S.B. Navathe, *Fundamentals of database systems* (Benjamin/Cummings, 1989).
[5] R.E. Fairley, *Software engineering concepts* (McGraw-Hill Book Company, 1985).
[6] R. Günthör, S. Jablonski, *Transaction-based application integration in workflow management systems*, Technical Report, University Stuttgart (1994); submitted for publication.
[7] J. Gray, A. Reuter, *Transaction processing: Concepts and techniques* (Morgan Kaufmann Publishers, Inc., 1993).
[8] E. Gamma, R. Helm, R. Johnson, R. Vlissides, *Design patterns: Elements of reusable object-oriented software* (Addison-Wesley Publishing Company, 1995).
[9] *SOMobjects: A practical introduction to SOM and DSOM*, Document number GG24-4357-00 (IBM Corporation, July 1994).
[10] *IBM FlowMark for OS/2*, Document number GH19-8215-01 (IBM Corporation, March 1994).
[11] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard, *Object-oriented software engineering: A use case driven approach* (Addison-Wesley Publishing Company, 1992).
[12] I. Jacobson, M. Ericsson, A. Jacobson, *The object advantage: Business process reengineering with object technology* (Addison-Wesley Publishing Company, 1995).
[13] F. Leymann, *A meta model to support the modelling and execution of processes*, Proc. 11th European Meeting on Cybernetics and Systems Research EMCR92 (Vienna, Austria, April 21-24, 1992), World Scientific 1992, 287 - 294.
[14] F. Leymann, W. Altenhuber, *Managing business processes as information resources*, IBM Systems Journal 33(2) (1994) 326 - 348.
[15] F. Leymann, D. Roller, *Business process management with FlowMark*, Proc. COMPCON Spring 94 (San Francisco, CA, February 28 - March 4, 1994) IEEE Computer Society Press 1994, 230 - 234.
[16] F. Leymann, *Supporting business transactions via partial backward recovery in workflow management systems*, Proc. BTW'95 Databases in Office, Engineering and Science, (Dresden, Germany, March 22-24, 1995), Springer 1995.
[17] F. Leymann, *Transaction concepts for workflow management systems* (in German), in: J.Becker, G.Vossen, Geschäftsprozeßmodellierung und Workflows (Thomson Publishing, 1995).
[18] D. Roller, F. Leymann, *Method and computer system for generating process management computer programs from process models* (Patent Application, January 1995).
[19] D. Roller, F. Leymann, *Methodology to generate C++ programs from Process Models with SOM Objects methods as activities*, (Patent Application, May 1995).
[20] Object Management Group, *The Common Object Request Broker: Architecture and Specification* (OMG, Framingham, MA, 1992).
[21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson, *Object oriented modelling and design* (Prentice Hall, Englewood Cliffs, 1991).
[22] D. Tkach, R. Puttick, *Object technology in application development* (Benjamin/Cummings, 1994).
[23] K. Walk, *Object oriented development of workflow management applications*, Technical report (IBM Austria, 1994).
[24] T.E. White, L. Fischer (ed.), *New tools for new times: The workflow paradigm* (Future Strategies Inc., Book Division, 1994).
[25] R. Wirfs-Brock, B. Wilkerson, L. Wiener, *Designing object oriented software* (Prentice-Hall, 1990).
[26] Workflow Management Coalition, *The workflow reference model*, Document Number TC00-1003, 1994.
[27] X/Open Guide, *Distributed Transaction Processing Reference Model (Version 2)*, X/Open Company Ltd., U.K., 1993.