

COO-FLOW: A PROCESS TECHNOLOGY TO SUPPORT COOPERATIVE PROCESSES

DANIELA GRIGORI

*Prism Laboratory, 45 avenue des Etats-Unis,
78035 Versailles Cedex, France
daniela.grigori@prism.uvsq.fr*

FRANCOIS CHAROY* and CLAUDE GODART†

*Université Henri Poincaré & INRIA, Campus Scientifique, BP 239,
54506 Vandoeuvre Cedex, France*

**francois.charoy@loria.fr*

†claudef.godart@loria.fr

In this paper we present a process management technology for the coordination of creative and large scale distributed processes. Our approach is the result of usage analysis in domains like Software Development, Architecture/Engineering/Construction, and e-Learning processes. The basic conclusions of these experiments are the following: (1) cooperative processes are described in the same way as production processes, but these descriptions are interpreted in a different way depending on the nature of the process, (2) the interpretation of process description depends mainly on the required flexibility of control flow and of data flow, and on the relationship between them, (3) the management of intermediate results is a central feature for supporting the cooperation inherent to these processes. COO-flow is a process technology that results from these studies. It is based on two complementing contributions: anticipation that allows succeeding activities to cooperate, and COO-transactions that allows parallel activities to cooperate. This paper introduces COO-flow characteristics, gives a (partial) formalization and briefly discusses its Web implementation.

Keywords: Workflow flexibility; cooperative processes modelling; coordination; cooperation.

1. Introduction

This paper describes a software contribution to the management of creative processes in co-design and/or co-engineering applications. The approach developed can be placed in the workflow initiative, but with particular attention paid to the flexibility requested by the nature of creative interactions. In other words, our objective is to develop a system that is as efficient for design processes as current workflow management systems are for administration and production processes. In addition,

it supports team work distributed on a large scale network, typically the Internet.

There have been many experiments on applying current workflow technology in design domain, but they resulted in a quick dismissal of this approach on working sites. This is due to the characteristics of creative processes that have been pointed out for a long time in the literature (long duration, uncertain development, important variability, interactivity, etc.) and that are badly supported by conventional workflow management systems. Yet, if an efficient support for process description and enactment is a strong request for design and engineering domains, the current usage is still often limited to project management tools. As a consequence, the management of the link between planning offices and working sites is quite weak and insufficient.

The approach developed in this paper is the result of a research including many usage analyses in domains like Architecture/Engineering/Construction (AEC), Software Development Processes, and Cooperative Learning. These studies lead us to the following assumptions:

- *in process books, creative and production processes are described in the same way* using boxes and arrows, but the interpretation of such descriptions changes depending on the nature of the application,
- in this interpretation, the *relationship established between control and data flow is determinant* and management of intermediate results (drafts, sketches ...) is a key point of this relationship,
- *process description and management* is only a contribution to creative project coordination *that must be combined with other contributing components* (object sharing, awareness, communication, etc.).

This paper describes the COO-flow model that results from these studies. It is based on two complementing contributions: anticipation allowing for succeeding activities to cooperate, and COO-transactions allowing for parallel activities to cooperate and for large scale distribution.

The rest of the paper is organized as follows. The next section motivates and gives overviews of our approach. Section 3 deals with cooperation between succeeding activities and Sec. 4 with cooperation between parallel activities; Sec. 5 discusses integration of both. Section 6 considers some implementation work. Section 7 presents related work and the last section discusses future directions and concludes.

2. Motivations and Overview

2.1. General considerations

The work described in this paper has been motivated by several experiments in co-design and co-engineering applications. A first application domain is software process. In this domain, software process modelling and enactment is considered as one of the more important elements for judging the capability of

a company to develop software (see levels of the Capability Maturity Model <http://www.sei.cmu.edu/cmm/cmm.html>). This request is even more important in the context of cross-organizational software development (we experimented in the AEE project (<http://aee.inria.fr/en/index.html>)).

We also found similar requirements in a project where the objective was to develop services to host virtual teams in AEC domain (for Small and Medium Enterprises). Even if current workflow technology was not efficient, we also understood the need for some process technology to support project management. But we also understood that this process technology would only be a contribution, a component of a larger infrastructure [12] (see also Sec. 10).

These motivations were confirmed by the interest taken by the Hitachi company in some of our preliminary work concerning cooperative transactions and their request to experiment the integration of their WorkCoordinator workflow management system with our transactional technology in order to model and enact cooperative processes. This experiment is related in Sec. 10.

During these experiments, we were convince of the possibility to develop a process technology for creative applications but with the following assumptions:

1. A process model for the coordination of a group of people implied in a creative process does not have to describe all the processes in full details, but only at a gross grain level;
2. In a real application, there is a lot of chance that there will not exist only one such process, but several. In addition these gross grain processes will have to co-habit with fine grain (traditional) processes specialized in well defined tasks;
3. Process management is only one contribution to cooperative process management that must co-habit with other dimensions of coordination like object-sharing and versioning, awareness providing, group decision support, etc.;
4. Creative processes modeling and enactment must be simple, as simple as traditional workflow modeling and enactment. Furthermore, we think that the modeling is the same, but that interpretation changes, depending on the nature of the application concerned.

COO-flow model has been inspired by these considerations.

2.2. Same description, but different interpretations

The basic idea comes from the following observation. Concretely, when we look at “paper descriptions” of processes, administrative or production processes on the one hand, and co-design or co-engineering processes on the other, we note that both classes are depicted in the same way using the same basic formalism: typically annotated boxes, arrows and so on.

However, it is clear that when we look at such a description, depending on the context in which it applies, we do not interpret it in the same way. For example, in an administrative context, we interpret two boxes separated by an arrow as a strict

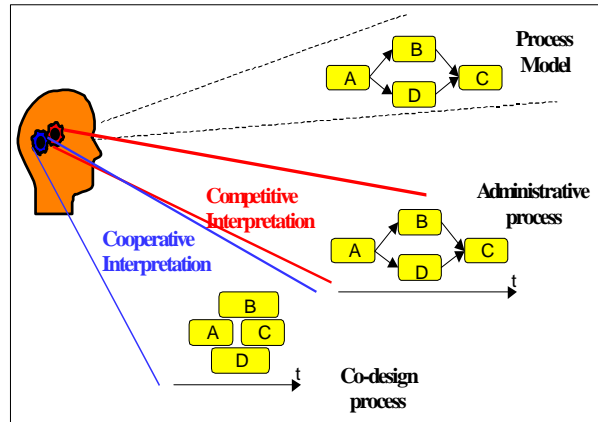


Fig. 1. Same description but different interpretations.

sequence of the two tasks represented by the two boxes and in our mind, the second task cannot start its execution before the first one has terminated. However, when we consider the same schema in the context of a co-design process, our interpretation is much more flexible. We know that this schema is a general indication, but that tasks will probably overlap, iterate. More generally, in the case of an administrative or production process, we see tasks as black boxes executing serially and sharing data only when they start and terminate. In the case of a creative process, we see tasks as white boxes making visible some results when executing, overlapping each other and even iterating their executions.

That is what Fig. 1 sketches. Starting from the same process model, and depending on the context, the mind interprets the same schema in different ways. As an example, making reference to Fig. 1, if we consider a typical administrative process and we take “*select a book*” for A, “*select a payment mode*” for B and “*verify payment capacity*” for C, we understand that these three activities will execute one after the other (see Fig. 1, competitive interpretation). If we consider a cooperative process and we take “*edit a plan*” for A, “*review a plan*” for B and “*validate plan*” for C, our mind infers the possibility for these activities to execute “cooperatively”. As an example, “*edit plan*”, “*review plan*”, and “*validate plan*” activity executions will probably overlap and share successive versions of the plan (see Fig. 1, cooperative interpretation).

2.3. *Difference of interpretation is matter of control flow flexibility and dataflow flexibility*

The question discussed here is: what makes the difference between a competitive interpretation and a cooperative interpretation? The difference is a matter of control flow and dataflow flexibility. A typical competitive process is a process where

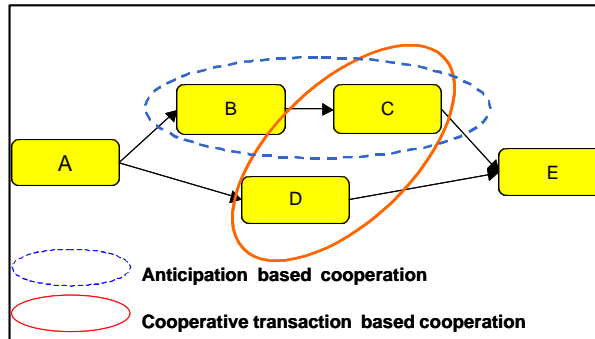


Fig. 2. Dimensions of flexibility.

activities are executed as ACID transactions^a, synchronized by typical workflow operators (operators of the WfMC model there are a good reference: sequence, and-join, and-split, or-join, or-split, etc.). Thus, relaxing the constraints of workflow operators and allowing activities to execute in a non isolated way will provide support for a flexible interpretation of process model.

More flexible is the control flow and the data flow, more cooperative is the interpretation. In COO-flow, flexibility is based on two complementing contributions: “anticipation” and “COO-transaction” (see Fig. 2).

Anticipation is the principle that allows an activity to start its execution even if all its conditions are not “completely” fulfilled. It contributes to control flow and data flow flexibility. This concept is developed in the next section.

COO-transactions allow activities to exchange (intermediate) results in a consistent way by providing data flow flexibility. This concept and its application is developed in Sec. 4.

2.4. Discussion

One question that arises at this point is: “as by definition creative activities are not isolated, why do we encapsulate them in transactions?” The response is that in most creative activities, if people interact by exchanging some “semi-consistent” data, nevertheless they want to be assured of a certain degree of security, of consistency, and with a minimum of programming. That is exactly the role of a transactional correctness criteria that defines some general consistency properties, the transactional protocol enforcing them in a transparent way.

In [2] authors demonstrate that the advanced transactions models, needed for modelling the advanced applications they were concerned with, could be modelled with basic workflow control structures and ACID transactions, or in other words

^aTraditional WFMS does not impose activities to be executed as ACID transactions, but we assume this restriction to support the consistency requested by our applications, in the vein of [2].

that workflow systems are a superset of advanced transaction models.

If this is theoretically true, our feeling is that the creative processes that we are concerned with cannot be modelled in the same way. This is due to the uncertain development (atomic activities are not initially known) and combinatory nature of creative process interactions [13].

To take into account these characteristics of creative processes, we developed the COO-transaction concept [29] that provides consistency guaranties in the presence of intermediate results exchange. A supplementary difficulty was the need for integration of this model with the ability to anticipate the execution of an activity.

3. Anticipation: Cooperation between Succeeding Activities

This section introduces the concept of “anticipation”. Intuitively, anticipation means that an activity can start its execution even if all its execution conditions are not fulfilled. Traditionally, an activity can start its execution only if control flow operators allow it (the preceding activity is terminated in case of a sequence, at least one of the preceding activities is terminated in case of an or-join, all the preceding activities are terminated in case of an and-join and so-on) and if data flow based conditions (traditional condition based on data values) allow it. On the contrary, anticipation allows an activity to start its execution even if these conditions are not yet fulfilled (the preceding activity is not terminated in case of a sequence, none of the preceding activities is terminated in case of an or-join, not all its preceding activities are terminated in case of an and-join and so-on) and if data flow based conditions are fulfilled, but only to a minimum. In practice, activities can start if they have input enough intermediate results, typically a configuration of drafts, that allows work to start.

Anticipation impacts the execution model of an activity in the following way. It is necessary:

- to introduce two new states in the activity model: *ready_to_anticipate* and *anticipating states*. Consequently, it is necessary to assign activities in *ready_to_anticipate* state to agents who are able to start their anticipated execution,
- to modify the process execution algorithm: while in traditional workflow, the only event that triggers a new step for electing executable activities is the *termination of an activity*, in the modified model, *activity start* and *data production* events can change the state of succeeding activities to the *ready to anticipate* state,
- to modify data flow to integrate data exchange between anticipated activities (to manage exchange of intermediate results).

3.1. Adding *ready_to_anticipate* and *anticipating states*

In traditional workflow, activity life cycle is roughly the following (see Fig. 3). When a process instance is created, all activities are in the *initial* state (except

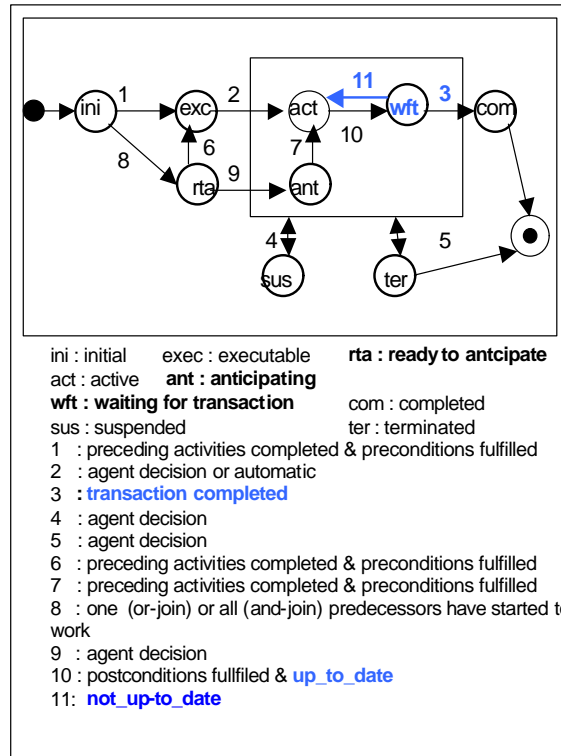


Fig. 3. Activity states.

the start activities that are directly *executable*). An activity enters the *executable* state as soon as the preceding ones have *completed* and its preconditions are fulfilled. As soon as an activity becomes executable, it is scheduled, i.e. it is assigned to all agents who actually qualify under its associated staff query. When chosen by an agent (or automatically), the activity becomes *active*. When its execution terminates normally, the activity reaches the *completed* state (final state). An *active* activity can be *suspended*, i.e. quiescent until returned to the executable state, or *terminated*, i.e. stopped before its normal completion without the possibility to return to the *executable* state.

Anticipation impacts this model as follows: if its anticipation conditions are fulfilled, an activity can enter the *ready to anticipate* state coming from the *initial* state. When entering the *executable* state, the activity is scheduled, i.e. is assigned to all agents who actually qualify under its associated staff query. A *ready to anticipate* activity can become, either executable if all its activation conditions are fulfilled with final results of preceding activities, or *anticipating* when one agent chooses the activity and starts the work (starts to anticipate). An activity in the *anticipating* state enters the *active* state when it is in a situation where it would

have been allowed to start its execution if it was not anticipating, in other words, when previous activities have completed.

3.2. *Conditions for anticipation*

We have analyzed several strategies. The three main ones are sketched below (and formalized in [15]):

1. *Free anticipation* — an activity in an *initial* state may anticipate at any moment. This allows activities to start their work earlier. In other words, control flow dependencies defined in the process model are interpreted at execution time as end-end dependencies; i.e. *an activity can finish its execution only when the preceding ones have*.
2. *Control flow dictated anticipation* — an activity may anticipate when, in the case of an *or-join* like operator, one of its predecessors has started to work, i.e. is in *anticipating* or *executing* state, and, in the case of an *and-join* like operator, all its predecessors have started their execution. With this strategy, the traditional start-end dependency between activities is relaxed, being replaced with a start-start dependency; i.e. *an activity can start its execution as soon as the predecessor has started*.
3. *Control flow and data flow dictated anticipation* — an activity can anticipate when one of its predecessors has started to work (is in *anticipating* or *executing* state) and for all its mandatory inputs, one intermediate result is available. This is like control flow anticipation with additional constraints concerning input existence. In addition, intermediate results can be requested to satisfy a minimal set of constraints.

3.3. *Formalization of anticipation strategies*

In this section we define the conditions for an activity to enter the “ready to anticipate” state for the control flow and data flow dictated strategies presented above. This specification extends the specification introduced in [22] for a traditional workflow. For a complete specification of COO-flow, see [15].

First, we need to introduce the definitions for *Activity state map* and *Data state map* [22], the applications that give the state of an activity (resp. a data element) at any moment during the execution of a process instance.

Definition 1. (*Activity state map*) Let N be the set of all activities of a process model and \mathbb{N} the set of natural numbers representing the time axis for each process instance ($0 \in \mathbb{N}$ represents the point in time when the associated instance of the process model is created). The activity state map

$$\omega : \mathbb{N} \times N \rightarrow S$$

associates at any point in time $i \in \mathbb{N}$ each activity A with its actual state $\omega(i, A)$.

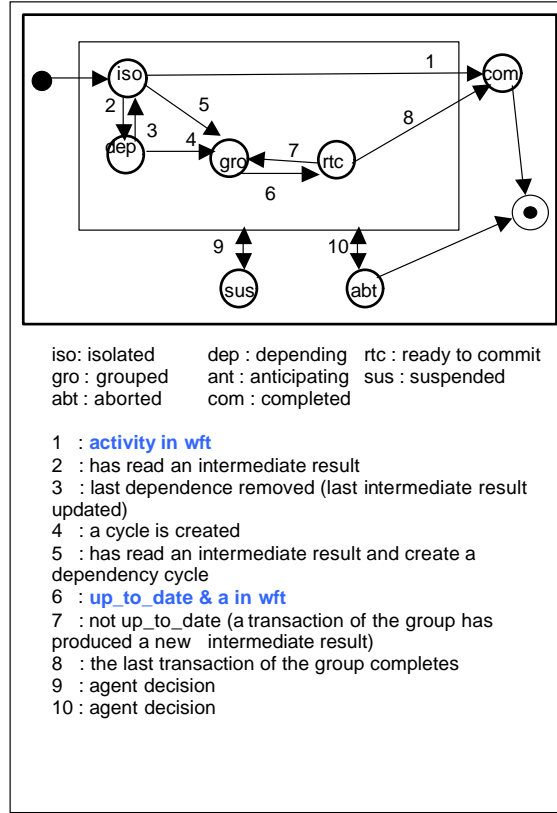


Fig. 4. Transaction states.

S includes all activity states relevant to execution: *initial*, *executable*, *active*, *ready to anticipate*, *anticipating*, *terminated*, *completed*, *suspended*.

Definition 2. (Data state map) Let V be the set of all process data elements (data needed as input by the activities, data required by conditions and the data to be exchanged between activities). The data state map

$$\delta : \mathbb{N} \times V \rightarrow S$$

associates at any point in time $i \in \mathbb{N}$ each data element v with its actual state $\delta(i, v)$.

The relevant states of a data element are *initial*, *intermediate* and *final*. An output data element of an activity enters the *intermediate* state when the activity executes a *Write* operation on this data; it enters the *final* state when activity completes.

Now we can define the conditions for an activity to enter the “ready to anticipate” state for the control flow and data flow dictated strategies presented in the

previous section (for the free anticipation strategy, *initial* and *ready to anticipate* state are merged).

Definition 3. (*Activities in State “Ready to Anticipate”, control flow strategy*) Activity A becomes *ready to anticipate* at time $i \Leftrightarrow$

1. $\omega(i-1, A) = \textit{initial}$
2. $\forall X \in A^\leftarrow, \omega(i, X) \in \{\textit{executing}, \textit{anticipating}\}$ if A is an AND-join node or a regular node
3. $\exists X \in A^\leftarrow, \omega(i, X) \in \{\textit{executing}, \textit{anticipating}\}$ if A is an OR-join node

where A^\leftarrow denotes the set of all direct predecessors of activity A.

A regular or an AND-Join activity in initial state enters the *ready to anticipate* state when its predecessors are activated for execution or anticipation (*executing* or *anticipating* state). An OR-Join activity enters the *ready to anticipate* state when one of its predecessors is activated for execution or anticipation (active or *anticipating* state).

Definition 4. (*Activities in State “Ready to Anticipate”, control and data flow strategy without conditions on data*) The control and data flow strategy imposes supplementary conditions concerning the availability of input data as follows. Activity A becomes *ready to anticipate* at time $i \Leftrightarrow$

1. condition 1 of Definition 3
2. condition 2 of Definition 3
3. $\forall v \in i(A) : (w, v) \in \Delta^{\text{in}}(A) \Rightarrow \delta(i, w) \neq \textit{initial}$ and $\exists v \in i(A) : (w, v) \in \Delta^{\text{in}}(A) \Rightarrow \delta(i-1, w) = \textit{initial}$

where $i(A)$ is the input container of activity A (the set of all its input data elements), (w, v) is a data connector connecting a data element w of the output container of an activity ($w \in o(X)$) with a data element v in the input of another activity ($v \in i(A)$), $\Delta^{\text{in}}(A)$ denotes the set of all data connectors (v, w) having as destination an element in the input container of activity A ($v \in i(A)$)

The third condition requires for all input elements ($v \in i(A)$) being destination of a data connector, the source of the data connector to be in *intermediate* or *final* state.

4. Cooperative Transactions: Cooperation between Parallel Activities

As introduced above, cooperation between parallel activities is mainly supported by COO-transactions. COO-transactions model relaxes the isolation property of ACID transactions in order to support consistent exchange of intermediate results between activities. For the sake of brevity, it is satisfying enough here to explain how its protocol provides an active support to cooperation while maintaining at the

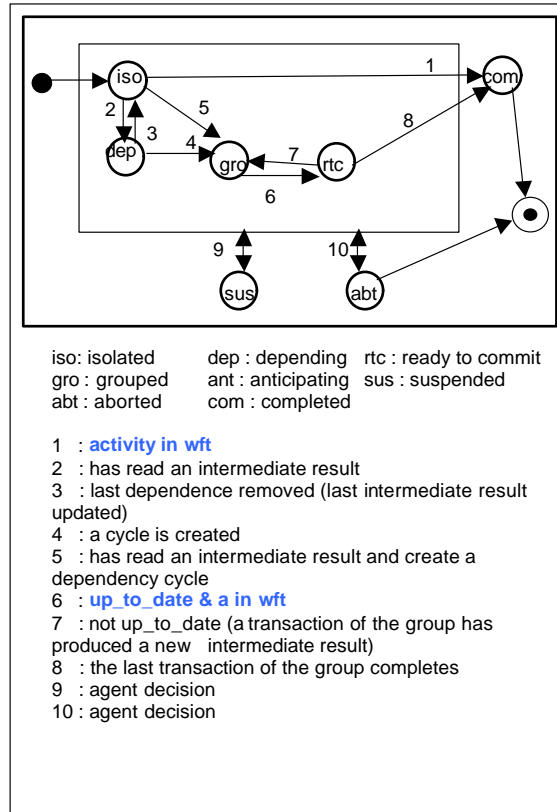


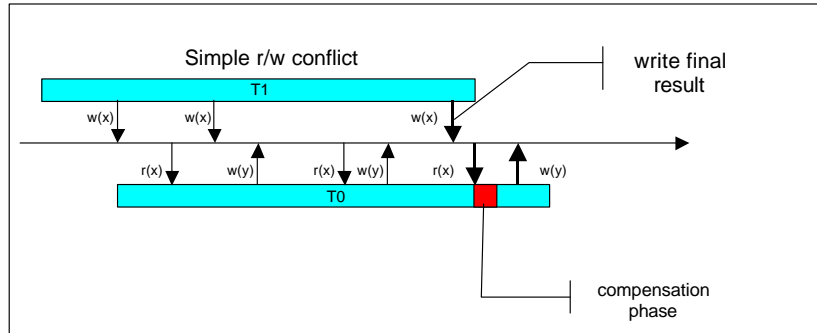
Fig. 5. Transaction states.

same time a sufficient level of consistency. Then we explain how process activities are encapsulated in COO-transactions.

4.1. COO protocol

The principle of COO-transactions is that a COO-transaction can read an intermediate result of another COO-transaction. A COO-transaction that has read an intermediate result of another transaction depends on it and must read the corresponding final result before to complete. COO-transactions implicated in a cycle of dependencies are grouped together and must agree on the final value of the shared data before to complete simultaneously.

A transaction that starts its execution enters the *isolated* state (called isolated with reference to ACID transaction executing in isolation), see Fig. 4. A cooperative transaction can read an intermediate result of another cooperative transaction. When a transaction T_0 reads an intermediate result (a result produced by a transaction during its execution before it completes) of an object x produced by a transaction T_1 , it becomes dependent on T_1 (see Fig. 5).

Fig. 6. T_0 uses intermediate results of T_1 .

When the transaction T_0 updates its value of x with the final value of x produced by T_1 , the dependency is removed. A transaction which has read one or several intermediate results, but that is not implicated in a cycle (T_0 is not depending on itself) is in the *dependent* state (see Fig. 6). A transaction which is the *dependent* state and which releases its last dependency enters the *isolated* state again.

Transactions involved in a cyclic dependency graph form a group of transactions: each transaction of the group enters the *grouped* state. A transaction cannot try to terminate if it is still dependent on another transaction (not *up_to_date*). If a group-member transaction tries to terminate (it is *up_to_date* with other transactions), and they are still transactions of the group in the *grouped* state, it enters the *ready to commit* (RTC) state. If a group-member transaction wants to terminate and the other transactions of the group are in the RTC state, all the transactions of the group terminate simultaneously, i.e. enter the *completed* state. However, if a group member produces a new intermediate result during the group termination phase, then this termination is aborted and all group-members re-enter the *grouped* state. A transaction can be *suspended* or forced to abort (*aborted* state). For a formalization of COO-transactions, see [5], [27].

The originality of the COO-transaction model [27], with regard to other advanced transaction models, is that it supports concurrent writing (including writing of two copies of the same document) while preserving some properties (COO-serializability). Especially this allows, thanks to intermediate results management, large scale process management, including disconnected work.

4.2. *Activities as COO transactions*

To support cooperation in a consistent way, each activity is encapsulated in a transaction. The problem is that a user can try to terminate an activity while the corresponding transaction is not *up_to_date*. To prevent this case, we have added in the activity model a new state, i.e. *waiting for transaction* (*wft*) and modified some transition conditions (see Figs. 3 and 4). The principle is that, when a user

wants to complete an activity, before this completion can effectively take place, the system must verify that this activity is consistent with the others, or in other words that the corresponding transaction can also complete. When an activity enters the *wft* state, i.e. its post-conditions are fulfilled, a command to complete is sent to the corresponding transaction; if the corresponding transaction can complete, the transaction and the activity pass simultaneously in the *completed* state. If the transaction is implied in a group, the activity has to wait in the *wft* state for the group termination (this will be triggered by the last transaction of the group wanting to terminate) and all activities of the group will complete simultaneously. If the transaction still has to work before to complete (for instance, it has not read the last final value present in the database, or a new result has been recently published by a group member), the activity re-enters the *active* state.

The introduction of a new activity state for the synchronization of activity and transaction termination, as introduced in this paper, is one possible solution. We studied another symmetric solution in the Corvette project where we extended the transaction model in a symmetric way. We prefer the solution deepened in this section for usage reasons, but in both cases, orthogonality of activity management and transaction management is not completely achieved (in the sense that we have to modify either the activity model or the transaction model to integrate them together).

Encapsulation of activities in COO-transactions allows the following supplementary types of data exchange:

- Data can be accessed on the user initiative from a shared workspace which control the access (following a pull principle).
- Activities can exchange intermediate results. Moreover, in order to support opportunist styles of work, activities situated in parallel braches can exchange data and even work concurrently on shared copies.

These types of data exchange are not possible in traditional workflow systems where activities are transactional black boxes and data is transmitted directly between activities (push mode).

New activity operations are introduced, *Write* and *Read*. These operations can be used by users (or even special tools) to manage publication of data during activity execution. *Write* operation publish a data element and makes it available to the other activities of the process instance. *Read* operation is used to read the current version of the data from the shared workspace.

5. Integration of Anticipation and COO-Transactions

In the previous sections, we have introduced anticipation and COO-transactions as complementary contributions to support flexible interpretation of creative processes. The problem we are concerned with in this section is that these two contributions are not completely orthogonal and each have an influence on the other.

Thus, we have pointed out two potential points of conflict:

- the termination of an activity (explained in the previous section),
- the risk of contradiction between the activity order implied by control flow and the activity order implied by data flow.

5.1. *Risk of contradiction between control flow and data flow*

Anticipation allows successive activities to execute partly in parallel, thus increasing the degree of parallelism. These activities can exchange data, generating dynamic dependencies between them. Although anticipation allows succeeding activities to run in parallel, the principle is that, at the end, things appear as if activities had not anticipated. Especially, if B succeeds A, A should start and terminate before B and exchanges that conduct to contradict such an order should be forbidden.

Reading intermediate results generates dependencies between activities dynamically; these dependencies are managed by the transaction manager. If A precedes B and A reads an intermediate result of B, directly or indirectly, the two activities will be grouped and forced to terminate simultaneously, what can be considered as contradicting the order relation defined by the control flow model. As example, in Fig. 7, A precedes B from the point of view of the workflow, and B has probably read an intermediate result of A, but A has also read an intermediate result of C that has read an intermediate result of B: A, B and C form a group of transactions.

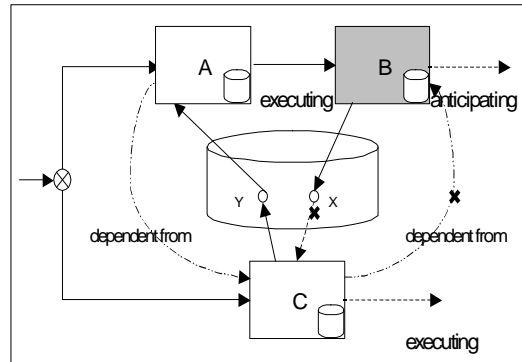


Fig. 7. Contradiction between control flow and data flow.

We think that in a large number of cases, the fact that activities which were defined as successive in the process model are forced to terminate synchronously by the transaction manager is not really a problem. However, if it is, this situation can be prevented by enforcing the following rule: *an activity cannot read an intermediate result of another activity if this induces a dynamic dependency between activities that contradict the flow dependency (static dependency) between them.* This can be

simply implemented as dependences are known and managed by the transaction manager.

6. Implementations

This section reports on two implementations related to COO-flow model. Both of them have been deployed in an Internet context, exploiting capabilities of intermediate results management.

The Corvette project <http://www.loria.fr/equipes/ecoo/corvette/> studied the feasibility of the COO-flow concept by the integration of the Hitachi WorkCoordinator WFMS with COO-transactions, one important design constraint of the project being the impossibility to modify WorkCoordinator code. As WorkCoordinator does not support *anticipation*, cooperation between succeeding activities was not feasible without changing the WFMS. Fortunately, we were able to demonstrate cooperation between parallel activities, with only some smooth intervention in the transaction model and a small intrusion in WorkCoordinator for defining triggers in its relevant database in order to capture important events (especially activity state changes). This experience is reported in [3].

We have implemented a new version of the COO-flow model by taking advantage of the *Enterprise Java Beans* technology and the *J2EE (Java™ 2 Platform, Enterprise Edition)* for security, transaction management, connection with databases systems, portability. This implementation uses the Jboss application server (<http://www.jboss.org/>). It is called Bonita and can be downloaded from (<http://woinville.loria.fr/nToxic/Project/bonita/Download/bonita-src.tar.gz>).

Also, Bonita is being tailored for different means in LibreSource (<http://www.libresource.org>), a project dedicated to distributed software development, and in Coopera (<http://woinville.loria.fr/coopera>), a project dedicated to e-learning for children. COO-flow prototype is being integrated also in ToxicFarm (<http://woinville.loria.fr>), a platform hosting services for distributed teams. ToxicFarm will be experimented in the context of cooperative learning and of software design and development at the scale of France.

7. Related Works

Recently, there have been much research on workflow flexibility [4, 20]. Based on the role and the use of process model, we distinguish three main approaches for flexibility.

The first approach considers *the process model as a resource for action* ([1, 23]). Basically, it means that the process is a guide for users upon which they can build their own plan. It is not a definitive constraint that has to be enforced. Thus, users have all the initiative to execute their activities. They are not constrained by the predefined order of activities but are inspired by it and encouraged to follow it. In this vein, [23] proposes to enhance the workflow model with goal activities and regions in order to allow its use as a resource for action. A goal node represents

a part of the procedure with an unstructured work specification; its description contains goals, intent or guidelines. If this approach provide a lot of flexibility on workflow execution, it also lack control on what effectively happens on working site (even if combined with user awareness).

The second approach uses the process as a constraint for the flow of work, but it is accepted that it may change during its lifetime. In other words, the process can be ally adapted during its execution. ADEPTflex [24], Chautauqua [9], WASA [30] and WIDE [6] provide explicit primitives to dynamically change running workflow instances. These primitives allow to add/delete tasks and to change control and data flow within a running workflow instance. Constraints are imposed on the modifications in order to guarantee the syntactic correctness of the resulting process instance. [28] [26] [10] propose methods to migrate instances from the old process definition to the new one without introducing errors.

The third approach for flexibility consists of evolving the process model itself to allow for more flexible execution. In this case, flexibility has to be modelled and is anticipated during the process modelling step. In Mobile [21], authors define several perspectives (functional, behavioural, informational, organizational, operational) for a workflow model, the definitions of perspectives being independent of one another. Descriptive modelling is defined as the possibility to omit irrelevant aspects in the definition of a perspective. In MCI [11], authors propose simple constructs that better represents real work patterns (optional activities, group activities, synchrony) to be used instead of a complex composition of elementary workflow operators.

The first two approaches consider flexibility by allowing to escape from the model itself. In one case, the model is a guide to reach a goal, in the other case, the model is a path to reach a goal that may change during its course. In the third approach, it is the model that evolves to provide the requested flexibility. The approach introduced in this paper is quite different. It is not based on the way the process model is used or instantiated, neither on the way it can be evolved or modelled: it adds flexibility in the workflow management system execution engine itself. This flexibility is based on the study of the relationships between control and data flow: anticipation of activity execution combined with exchange of intermediate results provide for a very effective support for cooperation between activities. In addition, as an activity can be prevented to anticipate and to publish intermediate results, different levels of flexibility can be defined and can co-habit in the same process.

Another possibility to provide a support for cooperative processes that was investigated in the literature is to merge workflow which supports formal and well-structured processes with groupware, which offers communication and cooperation tools for unstructured processes. [17, 29] integrates into a cooperative hypermedia system with many communication and cooperation features and process support capabilities. However, the correction of data exchange is not assured but is left to user responsibility.

During the past decade, traditional workflow systems we been extended with transactional features. Examples of these are TriGSflow [19] workflow system, which is based on an active database system using a multi-parent nested transaction model, and the Exotica [2] system, which shows how advanced transactions models, such as Linear Sagas and Flexible Transactions, can be implemented on top of existing workflow systems such as FlowMark. The basic ideas of other transactional workflows (ConTracts [25], WAMO [7, 8], WIDE [14], CREW [18]) are the transactional bracketing of parts of a process, attaching compensation and contingency activities to the activities of the process, declaring some of the activities to be vital (or critical), and defining points in the process up to which rollback occurs on failure, followed by forward execution. In these models, the cooperation is limited to passing results between activities in a programmed, predefined manner. They relax the atomicity property of a process, by dividing it in sub-transactions. However, these sub-transactions are still atomic. For these reasons, these models defined for a given application domain are difficult to adapt for cooperative processes that are dynamic, unpredictable and less structured.

8. Conclusion and Future Directions

This paper has introduced a new process model that fits very well the requirements of creative processes coordination by introducing control flow flexibility and dataflow flexibility without changing process description. Its technical feasibility has been demonstrated and its usability in distributed teams has been partially tested. In addition, this approach is not presented in opposition with existing ones, but on the contrary, as our experiments have demonstrated the need, as a complementary one.

As an example, COO-transactions can be introduced transparently as a set of cooperation patterns at the same level as MCI patterns [11], as we started to do in [12]. More generally, we understood during different experiments in different application domains that the approaches for workflow flexibility are complementary and that the better we would be able to make several of them co-habit, the better we would provide support for co-design and co-engineering processes. In fact, in a co-design and co-engineering application, it does not exist one single process but several ones. In addition, depending on the nature and granularity of processes, these processes follow different models. As a consequence, there is a need for a framework to reason about these different models and their interactions: this is our front line objective.

Anticipation strategies can be enhanced by adapting the algorithms developed in [16] to decide on the opportunity to anticipate an activity by calculating a probability for this activity to become active in the future.

Another objective is to study the relationships between process-based coordination and awareness based coordination. Especially, process knowledge can be used to provide awareness (process awareness) and other sources of awareness can

provide a good feedback of process acceptance on working sites on which process evolution can be based. This work is developed in the context of Toxic Farm (<http://wainville.loria.fr>). The same preoccupation exists in MCI [11].

In the continuation of this study, an ongoing objective is to make usage analysis in realistic situations by taking advantage of the applicative projects introduced above.

We are also concerned with the integration of the COO-flow model with other models, in the context of an intranet process or a multi-enterprises process.

References

1. A. Agostini and G. De Michelis, *Modeling the Document Flow within a Cooperative Process as a Resource for Action*, CTL-DSI, University of Milano, 1996.
2. G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthoer, and C. Mohan, "Advanced transaction models in workflow context", *Proc. 12th Int. Conf. on Data Engineering, ICDE*, New Orleans, 1996, pp. 574–581.
3. K. Baina, F. Charoy, C. Godart, D. Grigori, H. Skaf, S. Akifuji, T. Sakaguchi, T. Seki and M. Yoshioka, "Corvette: A cooperative workflow experiment", *3rd IFIP Working Conf. on Infrastructure for Virtual Enterprises*, Sesimbra, Portugal, 2002.
4. A. Bernstein, C. Dellarocas, and M. Klein, "Towards adaptive workflow systems (CSCW-98 workshop report)", *ACM SIGMOD Record* **28**(3) (1999) 518.
5. G. Canals, C. Godart, P. Molli, and M. Munier, "A criterion to enforce correctness of indirectly cooperating applications", *Information Sciences* **110** (1998) 279–302.
6. F. Casati, S. Ceri, B. Pernici and G. Pozzi, "Workflow evolution", in *Data and Knowledge Engineering*, Elsevier Science, January 1998.
7. J. Eder and W. Liebhart, "The workflow activity model WAMO", in *Proc. 3rd Int. Conf. on Cooperative Information Systems*, Vienna, Austria, 1995.
8. J. Eder and W. Liebhart, "Contributions to exception handling in workflow management", *Proc. 6th Int. Conf. on Extending Database Technology*, Valencia, Spain, 1998.
9. C. Ellis and C. Maltzahn, "Chautauqua workflow system", *30th Hawaii Int Conf. on System Sciences*, Information System Track, 1997.
10. A. Fent, H. Reiter and B. Freitag, "Design for change: Evolving workflow specifications in ULTRAFLOW", *CAISE 2002*, LNCS 2348, 2002, pp. 516–534.
11. D. Georgakopoulos, H. Shuster, A. Cichoki, D. Baker and M. Rusinkiewicz, *Collaborative Management Infrastructure: Final Report*, 2000, MCC.
12. C. Godart, C. Bouthier, P. Canalda, F. Charoy *et al.*, "Asynchronous coordination of virtual teams in creative applications: Requirements and design criteria", *Australian Conference Workshop on Information Technology for Virtual Enterprises*, ed. M. Orlowska, IEEE Press, 2001.
13. C. Godart, O. Perrin, and H. Skaf, "COO: A workflow operator to improve cooperation modeling in virtual processes", *9th Int. Workshop on Research Issues in Data Engineering Information Technology for Virtual Enterprises (RIDE VE'99)*, 1999.
14. P. Grefen, J. Vonk, E. Boertjes, P. Apers, "Two-layer Transaction management for Workflow Management applications", *Proc. 8th Int. Conf. on Database and Expert Systems Administration*, Toulouse, France, 1997.
15. D. Grigori, *Workflow Elements for Cooperative Processes Definition and Enactment*, PHD thesis in *Computer Sciences*, University Henri Poincaré Nancy1, Nancy, 2001, p. 135.

16. D. Grigori, F. Casati, U. Dayal and M. C. Shan, "Improving business process quality through exception understanding, prediction, and prevention", *27th Int. Conf. on Very Large Data Bases (VLDB)*, Roma, 2001.
17. J. M. Haake and W. Wang, "Flexible support for business processes: Extending cooperative hypermedia with process support", *Information and Software Technology* **41**, 6 (1999).
18. M. Kamath and K. Ramamritham, "Failure handling and coordinated execution of concurrent workflows", *Proc. 14th Int. Conf. on Data Engineering*, Orlando, Florida, February 1998.
19. G. Keppel, S. Rausch-Schott, W. Retschitzegger, and S. Vieweg, "TriGS: Making a passive object-oriented database system active", *JOOP7(4)* **63** (1994) 40–51.
20. M. Klein, C. Dellarocas and A. Bernstein (eds.), "Computer Supported Cooperative Work (CSCW)", *Journal of Collaborative Computing* **9** (2000).
21. S. Jablonski, "Mobile: A modular workflow model and architecture", *4th Int. Working Conf. on Dynamic Modeling and Information Systems*, Noordwijkerhout, NL, 1994.
22. F. Leymann and D. Roller, *Production Workflow*, Prentice Hall, 1999.
23. G. Nutt, "The evolution toward flexible workflow system", *Distributed Systems Engineering*, 1996.
24. M. Reichert and P. Dadam, "ADEPTflex — Supporting dynamic changes of workflows without losing control", *Journal of Intelligent Information Systems* **10** (1998).
25. A. Reuter, K. Schneider and F. Schwenkreis, *Contracts revisited*, in S. Jajodia and L. Kerschberg (eds.), *Advanced Transaction Models and Architectures*, Kluwer Academic Publishers, NY, 1997.
26. S. Rinderle, M. Reichert, and P. Dadam, "Evaluation of correctness criteria for dynamic workflow changes", *Proc. Int. Conf. on Business Process Management (BPM'03)*, Eindhoven, The Netherlands, 2003.
27. S. Tata, G. Canals and C. Godart, "Specifying interactions in cooperative applications", *11th Int. Conf. on Software Engineering and Knowledge Engineering, SEKE'99*, Kaiserslautern, Germany.
28. W. van der Aalst, "Exterminating the dynamic change bug. A concrete approach to support workflow change", *Information Systems Frontiers* **3**(3) (2001) 297–317.
29. W. Wang and M. J. Haake, "Flexible coordination with cooperative hypermedia", *Proc. ACM Hypertext'98*, 1998, pp. 245–255
30. M. Weske, "Flexible modeling and execution of workflow activities", *31st Hawaii Int. Conf. on System Sciences*, Software Technology Track (Vol VII), 1996.