

---

## Enhancing the flexibility of workflow execution by activity anticipation

---

Daniela Grigori\*

Université de Versailles St-Quentin en Yvelines,  
45, avenue des Etats-Unis,  
Versailles Cedex 78035, France  
E-mail: daniela.grigori@prism.uvsq.fr  
\*Corresponding author

François Charoy and Claude Godart

Université Henri Poincaré and INRIA,  
Campus Scientifique, BP 239,  
Vandoeuvre Cedex 54506, France  
E-mail: francois.charoy@loria.fr E-mail: claude.godart@loria.fr

**Abstract:** This paper introduces an evolution to classical workflow model that allows more flexible execution of processes while retaining its simplicity. Processes are described in the same way they are represented in design and engineering manuals, but the workflow system allows a flexible execution that is closer to the way participants actually execute their tasks in cooperative processes without computer support. This evolution is based on the concept of anticipation, that is, the weakening of strict sequential execution of activity sequences in workflows by enabling intermediate results to be used as preliminary input into succeeding activities. The concept can also be applied to provide rapid feedback to preceding activities and to improve parallelism between activities in the main process and its subprocesses. The architecture and implementation of a workflow execution engine prototype enabling anticipation is described.

**Keywords:** workflow flexibility; cooperative processes; codesign and coengineering processes; process modelling.

**Reference** to this paper should be made as follows: Grigori, D., Charoy, F. and Godart, C. (2006) 'Enhancing the flexibility of workflow execution by activity anticipation', *Int. J. Business Process Integration and Management*, Vol. 1, No. 3, pp.143–155.

**Biographical notes:** Daniela Grigori received her Master degree and PhD from the University Nancy I, France in 1998 and 2001, respectively. Since 2002, she has been an Assistant Professor at the University of Versailles, France. Her main research interests are in process modelling, business process intelligence, web services, coordination and cooperation models and systems.

François Charoy is an Assistant Professor at the University of Nancy 2. He received a PhD in Computer Science from the University Henri Poincaré of Nancy, France in 1992. His research interests include issues related to process modelling, workflow, long-term transactions and cooperative work. He has published papers in many conference proceedings and information systems journals. He has also contributed to important software development.

Claude Godart is a full-time Professor at University Henri Poincaré, Nancy 1 and Scientific Director of the INRIA ECOO project. His centre of interest concentrates on the consistency maintenance of the data mediating the cooperation between several partners. This encompasses advanced transaction models, user centric workflow and web services composition models. He has been involved in several transfer projects with industries (France, Europe, Japan) for a wide range of applications including e-commerce, software processes and e-learning.

---

### 1 Introduction

Current workflow models and systems are mainly concerned with the automation of administrative and production business processes. These processes coordinate

well-defined activities that execute in isolation, that is, synchronise only at their *start* and *terminate* states. If current workflow models and current workflow systems apply efficiently for this class of applications, they show their limits when one wants to model the subtlety of

cooperative interactions as they occur in interactive or creative processes, typically codesign and coengineering processes. Several research directions are investigated to provide environments that are more adaptable to user habits. These directions are described in Section 5. They propose most of the time complex evolutions of the basic workflow model. These evolutions may be hard to understand and to master by end users. Thus, our approach consists of adding flexibility to workflow execution with minimal changes of the workflow model. We try to reach this goal by relaxing the way the model is interpreted; users can take some initiative regarding the way they start the assigned activities, leaving the burden of consistency management to the execution engine.

In this paper, we introduce the idea of *anticipation* as a way to support more flexible execution of workflows. The principle is to enable an activity to start its execution even if all 'ideal' conditions for its execution are not fulfilled. Anticipation is very common in creative applications: reading a draft, starting to code without complete design and illustrate this idea. Anticipation adds flexibility to workflow execution in a way that cannot be modelled in advance. It can also be used to accelerate process execution as it increases parallelism in activity execution.

This work is part of the COO project, which aims at providing flexible support for cooperative processes in coengineering applications. We developed a cooperative workflow model and implemented a prototype that was used in different projects. This paper presents the flexibility of the control flow (the first results were presented in Grigori et al. (2001a)). The flexibility of data flow is presented in Grigori et al. (2004) and is based on the integration of the workflow system with a cooperative transaction manager.

This paper is organised as follows. In Section 2, we motivate our approach providing requirements for a cooperative workflow. In Section 3, we describe the evolution of the workflow execution engine to support anticipation and the constraints related to it in order to ensure consistent execution of a process. Section 4, is dedicated to the implementation of a workflow engine enabling anticipation. In Section 5, we present an overview of existing approaches for workflow flexibility. Finally, Section 6 concludes.

## 2 Requirements for codesign and coengineering applications

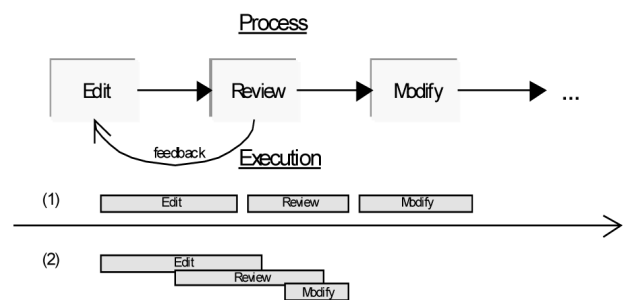
Traditional workflow management systems impose an end-start dependency between activities (Hollinsworth, 1995). This means that an activity can be started only after the preceding ones have completed. If this dependency is well suited for modelling production and administration processes, it is too restrictive for cooperative applications such as codesign and coengineering processes. This section introduces some requirements to support the flexibility necessary for the creative applications we consider. Then it enhances two upper level design criteria on which our approach is based.

*Need for anticipation:* most of the time, in a cooperative process activities overlap and start their work with some intermediate results (i.e. results produced by an activity during its execution, before its end, in opposition to final results, that are results produced by an activity when it completes) of preceding activities, even if all conditions for their execution are not completely fulfilled:

- preceding activities are not terminated (work on draft or partial results)
- activation conditions are currently evaluated as false or cannot be checked (some visa is missing, some tests have not been passed) and
- input data is not complete (but available data are sufficient to start working).

Allowing activities to start their execution in such conditions is what we call *anticipation*. Figure 1 depicts an execution of the same process without (1) and with anticipation (2). In the second case, the Edit activity provides an intermediate draft of the edited document to Review: Review and Modify activities can be started earlier. Thus, the whole process will probably terminate earlier.

**Figure 1** Execution without (1) and with (2) anticipation



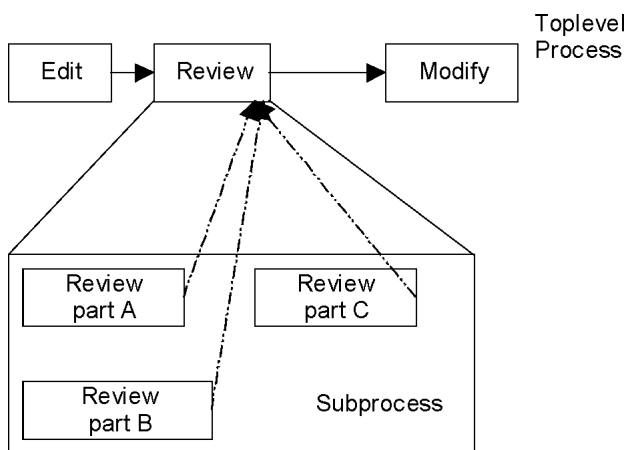
This type of execution has the goal to accelerate the process execution and to provide rapid feedback. The example of Figure 1 illustrates how the 'Review' activity can provide early comments while the 'Edit' activity is still running.

*Need to increase parallelism between subcontractors:* we illustrated how anticipation accelerates process execution. This is especially the case when an activity breaks down into one complex subprocess, whose activities are executed by different subcontractors acting on different subsets of the input dataset. Figure 2 illustrates this idea. The review of a composite document is on the responsibility of several actors depending on the document part and the actor role. In some cases, they can start their work as soon as they get their corresponding part and work in parallel.

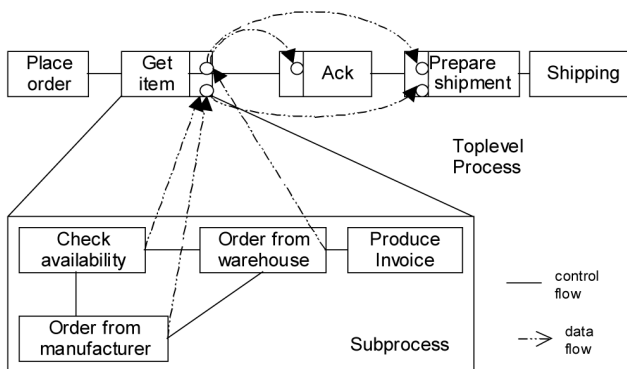
We can notice that this applies also for production applications as Figure 3, taken from (Hagen and Alonso, 1999) illustrates. The figure represents a typical process to handle the delivery of products by a retail company. The *Get item* activity takes care of the stock control and it is implemented as a subprocess which verifies if the item is available and otherwise orders it; finally an invoice is produced. The overall process, shown in the upper part of

the figure, starts with the initial placement of the order, followed by the *Get item* task explained above. After the products have been extracted from the warehouse, an acknowledgment is sent to the customer and the product is shipped. As the process is considered as a black box, the top level process waits the complete subprocess to finish before to proceed. This decreases the parallelism and increases the response time of the system. For instance, the acknowledgment to the customer could be sent as soon as it is known that the product is on stock. Similarly, it is not necessary that the shipping activity wait until the whole warehouse administration process concludes.

**Figure 2** Parallelism between subcontractors (Example 1)



**Figure 3** Parallelism between subcontractors (Example 2)



In Hagen and Alonso (1999) the parallelism between the top-level process and its subprocess is increased by using a supplementary type of control flow connector, an event-based control connector. This connector reacts to the occurrence of events and can be combined with the classical connector, which reacts only to the event corresponding to completion of activities. In Section 3, we explain how anticipation increases the parallelism without changing the process model and can be thus applied even to situation that were not foreseen at modelling time.

*Need for visibility of intermediate results:* “Intermediary objects produced during design activity are significant because they allow the coordination between designers and because they must be considered as real cognitive actors of design” (Grégori, 1999). In other words, intermediate results are first-order artefacts for supporting cooperation.

As introduced above, the main principle on which our vision is built is that people cooperate by exchanging drafts, sketches, intermediate results in order to increase efficiency of the virtual team work. The intermediate results allow to anticipate tasks, to obtain rapid feedback and a better group cohesion. Of course, intermediate results are progressively elaborated and a cooperation object will exist in several versions before the final value will be produced. To preserve enough privacy, the initiative to make an intermediate result visible is on the responsibility of human agents. This model is based on some usage analysis, including our proper experience in software development and discussions with designers in the domain of house building and car manufacturing.

*Need for a simple workflow model with variable coordination support:* cooperative processes are knowledge-intensive processes and they are mainly composed by activities executed by humans. The process model is the basic support for the awareness of process participants. For this reason, the model must be simple and its interpretation straightforward. Moreover, cooperative processes are unstructured and need dynamic modifications. A simple model facilitates user intervention for dynamic modifications.

A well defined workflow system should manage as well structured processes as cooperative processes. Several researchers (Dayal et al., 2001; Nutt, 1996) pointed out that workflow systems should be made more flexible and the goal of leading edge systems is to have variable coordination support being able to support structured and unstructured workflow (or some customised combination of both on a case-by-case basis).

### 2.1 Design criteria for codesign and coengineering workflow

*Flexible execution:* parallelism of execution between interacting activities is not really considered in traditional workflow systems and workflow engines. A way to surpass this limit is to change the workflow model, either by incorporating new synchronisation modes in the model or by allowing the model to be changed dynamically or both. Another way is to conserve the traditional workflow model Hollinsworth (1995) model can be taken as a good reference), but to change the workflow engine, by interpreting workflow scripts in a flexible way.

Based on the consideration that codesign, coengineering high-level process descriptions are done in the same way, with similar graphical representation as production and administrative processes, but require different degrees of flexibility for their execution, we think that the best approach is to conserve the traditional workflow model, but to change the workflow engine to increase parallelism between activities.

*Flexible data management:* while in traditional workflow models activities are ‘black boxes’ that are executed in isolation and produce all their output data at the their end, cooperative processes request more dynamic interactions between activities. An activity must be allowed to make visible its results while executing, that is,

before to terminate. The visibility of intermediate results is a key requirement for a flexible data exchange.

We focus here on these two aspects of flexibility required by codesign and coengineering applications. We do not address other design criteria, we refer the reader to (Dunn, 1990; Kazman et al., 1994) for examples, of other quality criteria common in the software engineering domain.

### 3 Flexibility of the execution model

In this section, we describe the evolution of the workflow execution engine to support flexible process execution and data exchanges and how we tackle the consistency problems that arise. In Section 3.1, the basic workflow model is presented.

#### 3.1 Basic model

The workflow model that we use is very simple, providing the basis for control and data flow modelling. We provide here a brief description that will allow us to explain the evolution we propose on the workflow engine.

The workflow model is based on process graphs. A process is represented as a directed graph, whose nodes are activities. Edges have associated transition conditions expressing the control flow dependencies between activities. Activities are units of work. Each activity has an associated staff query that specifies the resources that can execute it.

An activity having more than one incoming edge is a join activity; it has an associated join condition. For an *OR-Join* activity the associated join condition is the disjunction of conditions associated to incoming edges. For an *AND-Join* activity the associated join condition is the conjunction of conditions associated to incoming edges. An activity having more than one outgoing edge is a fork activity.

Activities have input data elements and produce output data elements. The circulation of data between activities is represented by edges between output elements of an activity and input data elements of another activity. The consumer must be a direct or transitive successor of the producer activity.

In summary, a process model is represented by a directed graph whose nodes are activities and whose edges represent control flow and data flow constraints.

#### 3.2 Anticipation

As introduced above, traditional workflow management systems impose an end-start dependency between activities (Hollinsworth, 1995). This means that an activity can be started only after the preceding ones have completed. However, in cooperative processes most of the time, successive activities overlap; activities start their work even if preceding activities are not yet finished.

Anticipation is the means we propose to support this natural way to execute activities while retaining the advantage of explicit coordination. Anticipation allows an

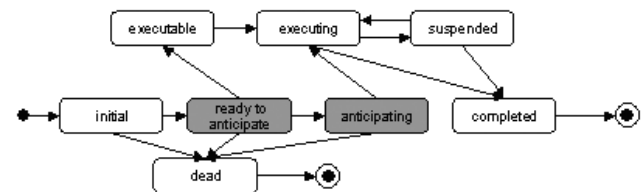
activity to start its execution earlier regarding the control flow defined in the process model. When preceding activities are completed and all activation conditions are met, an anticipating activity enters the normal executing state, that is, it continues its execution as if it never anticipated. At this time, final values of its input parameters are available. Having already been started, the activity is able to finish its execution earlier. The anticipation allows a more flexible execution, preserving, at the same time, the termination order of activities.

Example of Figure 1(2) is an execution with anticipation. The *Edit* activity provides an intermediate draft of the edited document to *Review*. In this case, *Review* and *Modify* activities can be started earlier. The whole process can thus be finished earlier.

#### 3.3 Execution states of an activity

In order to support the anticipation of activities, the workflow execution model has to be adapted. Our approach is to extend the traditional model (we start from the model defined in Leymann and Roller (1999)) to take into account anticipation. Two new activity states are added: *ready to anticipate* and *anticipating*. The *ready to anticipate* state indicates that the activity can start to anticipate. When an agent having the adequate role chooses it from its *to do list*, the activity enters the *anticipating state*. Figure 4 depicts a state transition diagram including the new added states. Note that before completion, even if an activity has started to anticipate, it must pass through executing state.

Figure 4 State transition diagram for activities



When a process is started, all activities are in *initial* state, except the start activities (activities with no incoming edges), that are in *executable* state.

*Transition from initial to ready to anticipate state:* concerning the moment when an activity in initial state can start to anticipate, several strategies can be considered:

- 1 *Free anticipation:* an activity in *initial* state may anticipate at any moment (*ready to anticipate* state merged with *initial* state). 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; that is, an activity can finish its execution only when the preceding one has. In our example, it would mean that *Modify* could start at any time. Free anticipation should be reserved to very special cases.
- 2 *Control flow dictated anticipation:* an activity may anticipate when its direct predecessor has started to work, that is, is in *anticipating* or *executing* state. For

an *OR-join* activity, at least one of its predecessors must be in the *anticipating* or *executing* state. For an *AND-Join* activity, all the preceding activities must have been started. In this case, the traditional start-end dependency between activities is relaxed, being replaced with a start-start dependency, that is, the successor activity can start its execution as soon as the predecessor has started. In our example, that means that *Modify* can be started as soon as *Review* has been started. As we explained before, the end-end dependency is ensured for all the anticipation strategies.

- 3 *Control flow and data flow dictated anticipation*: an activity can anticipate when its predecessors are in *anticipating*, *executing* or *completed* state and for all its mandatory inputs, there are values available. This requirement ensures that the anticipating activity has all its input elements available and its predecessors have already started to work. This is like control flow anticipation with additional constraints concerning inputs existence. In this case, the *Modify* activity could start only with a draft document and some early versions of the comments. The user in charge can see what he will have to do and start processing some comments. Note that in this case some synchronisation will have to be done with the final version of the document.

These three strategies have an impact on the general flexibility of the process execution. While the first one is very open but can lead to a lot of inconsistencies, the last one is more rigid and remove most of the interest of anticipation. In the rest of this paper, we consider that the implemented policy is the second one.

*Transition from ready to anticipate to anticipating*: as soon as an activity becomes *ready to anticipate*, it is scheduled, that is, it is assigned to all agents who actually qualify under its associated staff query. It is important to note that users know the state of the activity and can decide to anticipate. When one of these agents starts to anticipate, the activity passes in *anticipating* state and disappears from the *to do list* of the other agents.

*Transition from anticipating to executing*: an activity in *anticipating* state passes to *executing* state when it is in a situation where it would be allowed to start its execution if it was not anticipating; in other words, when previous activities have finished. Especially, it means that for all intermediate results it has used, it has got the corresponding final result. The user executing the task will have to do the corresponding update.

*Transition from ready to anticipate to executable*: an activity in *ready to anticipate* state passes to *executable* state in the same conditions as traditionally an activity passes from *initial* to *executable*. This situation corresponds to an activity that did not anticipate.

*Transition from anticipating to dead*: an activity in *anticipating* state passes to *dead* state if it is situated in a path that is not followed in the current workflow instance. Such a transition has the same motivations as for a traditional workflow activity to go from the *initial* state to the *dead* state. An anticipating activity makes the

hypothesis that it will be executed. This is not sure. Thus, it is possible that the work executed by it become useless. However, we think that, due to the nature of the applications we consider, this situation will not occur frequently: the objective of anticipation is to make the right decision at the right time thanks to rapid feedback.

We can see from this description, that modifications of the workflow execution engine to provide anticipation are not very important. 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 navigation 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 exchanges* between anticipated activities (to manage exchange of intermediate results). In order to gain all the benefits of anticipation of activity execution, it is necessary to allow also early circulation of data between activities, that is, publication of early or intermediate results in the output container of executing activities.

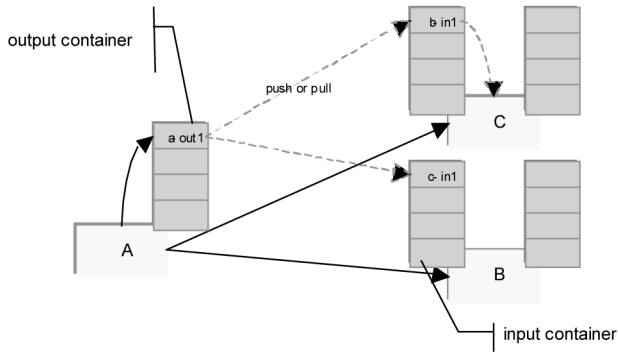
The last point (modification of data flow) is detailed in Section 3.4.

### 3.4 Data flow supporting intermediate results

As we consider mainly interactive activities, the user can decide to provide output data before their end and possibly with several successive versions. Two 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 updates an output element and makes it available to succeeding activities.

Consider the example of Figure 5 where activity *A* invokes the operation *Write (aout<sub>i</sub>)* to publish its output data *aout<sub>1</sub>*. Suppose that in the data flow definition, two edges exist that have *aout<sub>i</sub>* as origin: an edge (*aout<sub>i</sub>*, *bin<sub>i</sub>*) between *A* and *B* activity and another one, (*aout<sub>i</sub>*, *cin<sub>i</sub>*) between *A* and *C* activity. If *B* or *C* is in anticipating state, they must be notified about the existence of new data (pull mode) or notified of arrival of data in their input container (push mode).

*Read* operation is used by anticipating activities in pull mode to update an input data with the new version published by the preceding activity. For unstructured data, a mechanism must be provided to synchronise with new versions (merge for instance). For text files, this is a common feature supported by version management systems.

**Figure 5** Data flow between an executing activity and anticipating activities

Activities are no more isolated in black box transactions. They can provide results that can be used by succeeding activities. If the succeeding activities are interactive activities, the users in charge can consider taking this new value into account. They may also choose to wait for a more stable value supposed to arrive at a later time. Breaking activity isolation is necessary in order to benefit from the ability to anticipate but it may also cause some problems of inconsistency. These problems and the way to address them will be described in Section 3.8.

### 3.5 Formal definition of anticipation

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 Leymann and Roller (1999) for a traditional workflow. For a complete specification of COO-flow (see Grigori, 2001).

Firstly, we need to introduce the definitions for *Activity state map* and *Data state map* (Leymann and Roller, 1999), the applications that give the state of an activity (respectively 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  $\mathcal{N}$  the set of natural numbers representing the time axis for each process instance ( $0 \in \mathcal{N}$  represents the point in time when the associated instance of the process model is created). The activity state map

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

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

$S$  includes all activity states relevant to execution: initial, executable, active, ready to anticipate, anticipating, terminated, completed and 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

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

associates at any point in time  $i \in \mathcal{N}$  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 Section 3.4 (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  $i \Leftrightarrow$

$$1 \quad \omega(i-1, A) = \text{initial}$$

$$2 \quad \forall X \in A^{\leftarrow}, \omega(i, X) \in \{\text{executing}, \text{anticipating}\} \text{ if } A \text{ is an AND-Join node or a regular node}$$

$$\exists X \in A^{\leftarrow}, \omega(i, X) \in \{\text{executing}, \text{anticipating}\} \text{ if } A \text{ is an OR-Join node where } A^{\leftarrow} \text{ denotes the set of all direct predecessors of activity } A.$$

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  $i \Leftrightarrow$

$$1 \quad \text{condition 1 of Definition 3}$$

$$2 \quad \text{condition 2 of Definition 3}$$

$$3 \quad \forall v \in i(A): (w, v) \in \Delta^{\text{in}}(A) \Rightarrow \delta(i, w) \neq \text{initial} \text{ and } \exists v \in i(A): (w, v) \in \Delta^{\text{in}}(A) \Rightarrow \delta(i-1, w) = \text{initial}$$

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.

### 3.6 Anticipation and rapid feedback

Besides supporting early start, anticipation can be used to provide rapid feedback to preceding activities. A communication channel can be created backward of the defined flow of activities. As anticipation allows successive activities to execute partly in parallel, it is natural to imagine that people may have some direct feedback (e.g. comments) to provide to user of preceding activities.

The example of Figure 1 is the most obvious one: the *Review* activity can provide early comments while the *Edit* activity is still running. In this case, the *Review* activity may last longer. It will have to double check for the comments that may have been used by the *Edit* activity and remove them from its list. The *Modify* activity is shorter: some comments have been processed during the preceding phase.

### 3.7 Anticipation to increase parallelism between a process and its subprocess

In a traditional workflow, an activity is a black box that produces output data at the end of its execution. In our approach, an activity may fill an output parameter in its output container as soon as it is produced. These partial results become available to succeeding activities. They can enter anticipating state and initiate actions based on these results. For instance, let us consider the activity of sending a letter to a customer; the letter can be prepared in anticipating state with available data and really sent out in executing state. In this way, the process execution is accelerated.

Anticipation is especially useful when the activity is a subprocess. The output container of the activity is the output container of the process implementing it. Similar to an activity, a process can gradually fill data produced by its activities in its output container. These data become available for subsequent activities; otherwise they would have to wait the end of the subprocess. Anticipation increases the parallelism between the main process and the subprocess implementing the activity.

To illustrate this, we can use the example presented in Section 2 depicted in Figure 3. The output data of the subprocess are the warehouse where the product is available and the delivery date. The data are filled in the output container of the subprocess as soon as its corresponding activities produce them; succeeding activities in the top-level process can enter anticipating state.

As soon as the product is in stock or the date when it will be received from manufacturer is known, these data are filled in the output container of the subprocess and made available to succeeding activities. The acknowledgement to the customer can be prepared before the termination of the subprocess. Similarly, the *Prepare Shipment* is initiated as soon as it is known from which warehouse the product will be delivered and at what date it will be available.

For this kind of process, control and data flow dictated anticipation can be applied efficiently. However, the *Shipping* activity cannot be anticipated. This is a typical activity that can be executed only when all the preceding activities are finished.

### 3.8 Synchronisation and recovery

Visibility of intermediate results is necessary in order to benefit from the ability to anticipate and to support cooperative work. However, an activity that has published

results during its execution may fail or die. The problem is *how to compensate* the visibility of such results (it can be related to *dirty read* problem in traditional concurrency theory). This problem is tackled by applying the following rules:

An activity that has read an intermediate result must read the corresponding final result if necessary (it is different from the intermediate result).

*Synchronisation with previous activities*: an activity can enter completed state only from executing state. So, an anticipating activity, at a given moment will enter executing state. At this moment, previous activities will be completed and their final results available. When the activity enters executing state, its input container will be actualised with the final versions of its input parameters.

*Synchronisation in case of feedback*: consider an example presented in Figure 1. If state transition assures that the *Review* activity will read the last version of the *Edit* activity, it does not assure that the last result of the comment will be read by the first activity. Based on the principle that users are aware of risks introduced by intermediate results, we make the hypothesis that the management of intermediate results for preceding activities (feedback purpose) can be completely let under the responsibility of user. It does not need to be explicitly managed by the workflow system.

However, if feedback is important for the process, explicit loops can be defined in the process model. In the workflow model we base our work on, the repetition of a group of activities is specified by defining them as a block activity; the exit condition of this block activity then represents the proper condition for when to leave the loop.

In our example, *Edit* and *Review* activities would form a block activity. If the last comments were not taken into account, the exit condition of the block activity would not be fulfilled and the process will be re-instantiated; the *Edit* activity would finally take into account the last comments.

An activity cannot produce intermediate results if it is not currently 'certain' that it will enter the normal executing state. Independently of the anticipation strategy, we adopt a conservative approach concerning the moment when an anticipating activity may publish data.

For this purpose, only anticipating activities satisfying the following conditions can publish anticipated results:

- if the activity is a regular node (activity having one incoming edge), the predecessor must be in executing state
- if the activity is an *OR-Join* activity, it must exist a predecessor in the executing state and
- if the activity is an *AND-Join* activity, predecessor activities must be in executing or completed state.

These conditions ensure that this activity has a good chance to terminate, that is, it should enter the executing state and not pass in *dead* state (because of the control flow). Of course, this is an optimistic approach: nothing ensures that the preceding activities will not be cancelled neither that a human agent will not kill them.

In case where an executing activity is cancelled, anticipating activities that published data may need to be compensated. As only direct successors could publish data, the influence of cancelling an activity is limited. After the state changes, the status of anticipating activities is recomputed. They may go to dead state; the work done in anticipating state is lost. Otherwise, they remain in anticipating state. New updated values will be provided by the execution of their preceding activities. They will have to resynchronise with the next valid input.

As we can see, anticipation does not have an important impact on the general problem of workflow recovery, as long as activities do not have side effects. The work done in anticipating state may be lost but this is the price of flexibility. In case of activities having side effects outside of the scope of the workflow system, we are still obliged to introduce a special case described in Section 3.9.

### 3.9 Suitability and applicability of anticipation

Anticipation is not suitable in any situation and not applicable for any activity type. For instance, it cannot be applied for an automatic activity having effects that cannot be compensated. However, it can be applied for automatic activities that are of retry type (flexible transactions (Zhang et al., 1994)). For example, an activity that searches available flight tickets in a database, can be automatically restarted at each modification of its inputs if the execution speed of the process is more important for the application than the overload created in the external database. A similar example is an activity that compiles a program at each modification of one of its modules.

During the definition phase of the process, it may not be possible to know in advance when users will need to anticipate but it is possible to know which activities must not anticipate. These activities will be marked as such and will follow the classic execution model, that is, the start-end type of dependency will be enforced.

For the anticipation of an activity located on a conditional branch, an useful information for user in order to decide on the opportunity to anticipate and to avoid lost work is the probability that activity is carried out in the current instance (probability that the condition associated with the branch is evaluated as true). Simple queries applied to the workflow history database can be used to calculate the percentage of the number of instances when the activity was executed from the total number of instances. By applying the techniques presented in Grigori et al. (2001) it is possible to obtain better estimates of the probability of execution of an activity (for example, by discovering strong correlations with the initial data of the process, data related to the execution of the preceding branches, etc.).

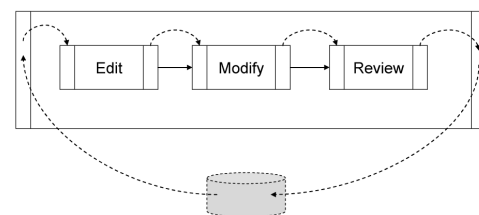
The anticipation strategy for the process execution is decided when the process is instantiated and can be changed from one instance to the other. Depending on the level of flexibility required, free anticipation, control flow dictated anticipation or data flow dictated anticipation will be used during all the process. Usability studies are required here to know precisely the actual impact of each strategy.

### 3.10 Identifying potentials for anticipation and designing processes

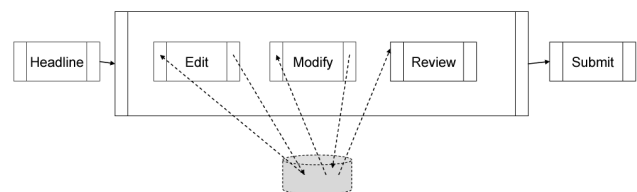
The anticipation strategy and consequently the flexibility of execution can be adapted according to the process needs and according to the work practices of the group involved in the cooperative process.

In extremis, a traditional (or competitive) process is simply a cooperative process where anticipation and the publication of intermediate results are prohibited by the execution engine. We consider again the process of editing a document to illustrate the various possible execution types. The strictest execution type is the execution that we call competitive, presented in Figure 6. When the process is initiated, the document is copied from the shared workspace and it is transferred from one activity to the following one, activities being executed in a strict order. The most flexible execution can be obtained by defining process fragments or even isolated activities and by omitting the definition of data flow. In this case, the activities read data from the shared workspace and the correction of data exchanges is ensured by a cooperative transaction manager (Grigori et al., 2001b). The role of the workflow system is limited to ensure the assignment of activities to participants and the control flow defined in process fragments. The unstructured parts can be included in a more structured process, as in the example of Figure 7.

**Figure 6** Different degrees of flexibility – strict execution



**Figure 7** Different degrees of flexibility – ad hoc cooperation



Between the two extremes, strict execution and the ad hoc cooperation, various anticipation strategies and various data exchanges modes can be defined to meet the needs of the cooperative applications.

Our cooperative workflow model allows a flexible execution and a flexible data exchange. We think that it facilitates the initial acceptance of the workflow system. Once the system is in place and a significant number of instances is carried out, the analyse of process logs using techniques presented in Grigori et al. (2001) allows to understand the way in which people work, why and when they anticipate, how they exchange data. This analysis can also reveal process patterns which appear regularly. These patterns could be modelled by using new operators (for example, COO operators Godart et al. (1999)) or new

dependences which represent better the work practices represented by the discovered patterns.

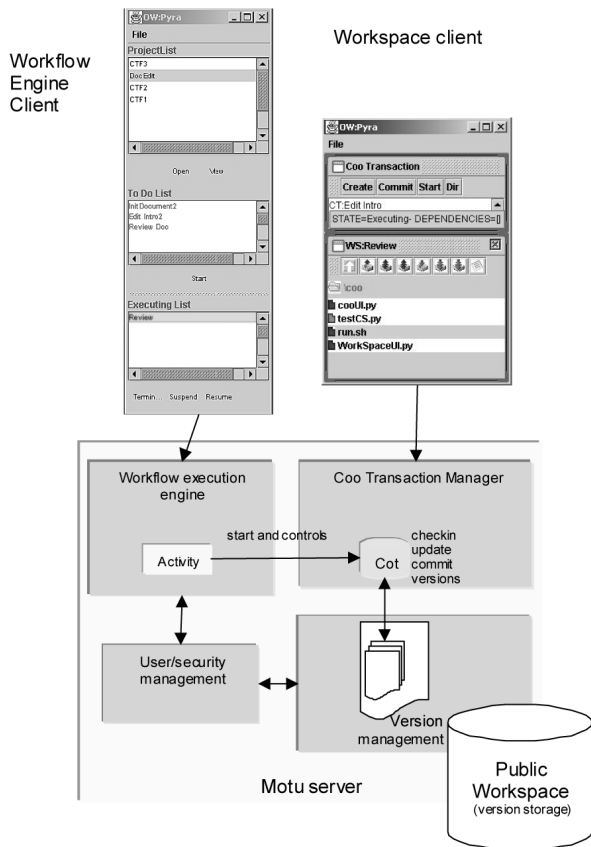
In this section, we have described how a simple modification of the classical model can enhance the flexibility of workflow execution in different ways. Now we are going to present how it is integrated in a larger framework to support cooperative workflow execution.

### 4 Implementation

We implemented a workflow execution engine enabling anticipation as part of the MOTU prototype (Godart et al., 2004) whose goal is to provide a framework to support cooperative work of virtual teams. The prototype is written in Java (motu.sourceforge.net).

The basis for the implementation is a non-linear version management server that provides the functionalities for public and private workspace management. This system has been extended with a basic workflow execution engine that implements anticipation (see Figure 8). Another component of the system is a cooperative transaction manager that allows exchanges of data between concurrently executing activities. Users are notified about the actions of other group members in their related projects: we consider that process and data awareness are important features for a cooperative environment.

Figure 8 MOTU basic architecture



Anticipation is implemented as part of the Workflow execution engine. Activities that are not anticipatable can be specified. A Coo Transaction is associated to each

activity (Grigori et al., 2001b, 2003). This Coo transaction provides support for optimistic concurrency control between activities. A Coo Transaction (Canals et al., 1998) has a private base (a workspace) that contains a copy of all the objects accessed (read and updated) by activities; it also serves as communication channel between successive activities.

Figure 9 shows the actual interface for three different users executing the process we used as example. The first one (V4) is actually executing the *Edit* operation. The second one (Pyra) is anticipating and has started the *Review* activity. *Review* appears in his executing list (list of activities that user started to execute) with a different colour. Data exchanges are done through their private workspace. V4 can provide Pyra with drafts of its document. Pyra can transmit comments (feedback) using the same channel. In this case, Vortex could start to modify the document. *Modify* is in its work list as an anticipatable activity. The anticipation strategy for the process is the control flow dictated anticipation.

We have implemented a new version of the workflow model using *Enterprise Java Beans* and the *J2EE (Java™ 2 Platform, Enterprise Edition)*, taking advantage of their features for security, transaction management, connection with databases systems and portability. This implementation uses the Jonas application server (<http://www.objectweb.org/>). It is called Bonita and can be downloaded from (<http://forge.objectweb.org/projects/bonita/>). Bonita provides an adaptative engine that supports different kind of strategies for anticipation. It also supports process evolution at runtime and most classical features of WFMS. More information on Bonita can be found at <http://bonita.objectweb.org/>.

Also Bonita is being tailored for different purposes in LibreSource (<http://www.libresource.org/>), a project dedicated to distributed software development, and in Coopera (<http://coopera.loria.fr/>), 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 has been experimented in the context of cooperative learning and of software design and development at the scale of France.

### 5 Related work

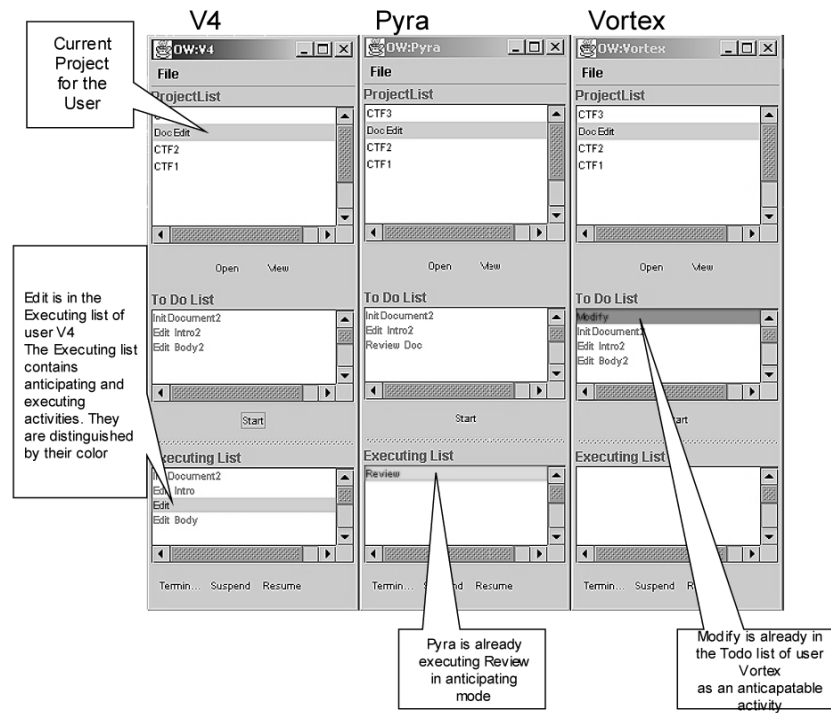
In this section, we discuss the relationship of our work to process modelling and to different approaches for workflow flexibility.

#### 5.1 Process modelling

We present briefly different techniques that have been used for modelling processes. For a survey on languages for modelling workflows, (see Weske and Vossen, 1998).

One of the most used techniques for modelling and analysing workflows is Petri Nets. A process model is represented by a PN by mapping activities to transitions and instances to tokens. By using high-level Petri nets, extensions such as colours and hierarchical PN provide

Figure 9 Actual execution of the example process



capabilities to represent workflow data, conditions and hierarchical structure of a process model. A particular type of Petri nets, called Workflow Net (WF-net) has been defined to specifically model workflow (van der Aalst, 1998). WF-nets allows proof of correctness (guarantees that the workflow will execute without deadlocks and that it will terminate eventually) and techniques for performance analysis. Other examples of Petri-net models tailored towards the requirements of workflow modelling are FUNSOFT (Gruhn, 1995) and Flow Nets (Ellis et al., 1995).

Unified Modelling Language (UML) diagrams have been also proposed to represent processes, but this approach has some limits. Bruno et al. (2002) pointed out that no single type of diagram captures all of the information needed to describe a process. Moreover, there is no clear semantics regarding the internal correspondence between diagrams used in the same model. Bastos et al. (2002) extended UML activity diagrams to include all the information needed for modelling a process. Wirtz et al. (2001) propose an extension of UML by using OCoN, a high-level Petri-net formalism for modelling dynamic behaviour. The goal of the approach is to specify all the relevant aspects of modelling and developing workflow systems in an integrated design language based on an object-oriented methodology.

In the area of distributed computing, a set of rigorous mathematically founded approaches have been developed, among which process algebras to formally define concurrently executing processes and their communication behaviour. These approaches (CCS Milner, 1980 and CSP Hoare, 1985) focus mainly on formal properties of distributed computations. Organisational and technical (for instance data flow) aspects, which are important for workflow, cannot be represented in these approaches.

State and activity charts were developed by Harel (1988) to specify the behaviour of reactive technical systems. They are dual view of a specification. Activity charts describe the functional decomposition of the modelled system in activities (the active components) and the data flow between activities. State charts specify the control flow between activities. The Mentor project (Wodtke et al., 1996) used this formalism to model workflows. The advantage of the state chart formalism is that it provides techniques and tools to formally prove properties of the modelled system, but the separation of control flow and data flow in state and activity charts may make the modelling process not easily understandable by application domain experts.

The most part of workflow systems and prototypes use specific and often proprietary graph-based languages. These languages allow to specify workflow activities, their hierarchical relationship and their control flow and data flow constraints using directed graphs. These graphs are enhanced to cover the workflow aspects presented above. Some examples of graph-based languages are Flowmark (Leymann and Altenhuber, 1994), Adept (Reichert and Dadam, 1998) and the model proposed by Hollinsworth (1995).

As we explained in Section 2, one of the requirements of cooperative processes is to have a simple workflow model, allowing a graphical representation. The graph-based language satisfies this criterion. Moreover, as this is the model described by the WFMC and used by most part of workflow systems and prototypes, our approach can be easily applied to existing systems.

In all the approaches for modelling processes presented above the dependency between activities is start-end dependency, that is an activity can start only after precedent activity has terminated. Data flow is either not considered or too restrictive.

In the following, we discuss different approaches for workflow flexibility.

## 5.2 Workflow flexibility

Workflow flexibility has been studied extensively as a mean to improve workflow management systems acceptance (Klein et al., 2000). Based on the role and the use of process model, we distinguish three main approaches for flexibility.

The first approach considers the process as a 'resource for action' (Suchmann, 1987). 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 keep 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.

Nutt (1996) propose 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. Agostini and De Michelis (1996) argue that a plan as a resource for action must support users awareness, helping them to situate in the context of the process, either to execute it or to escape to it in order to solve a breakdown.

The second approach uses the process as a constraint for the flow of work, but it is admitted that it may change during its lifetime. The process can be dynamically adapted during its execution. ADEPTflex (Reichert and Dadam, 1998), Chautauqua (Ellis and Maltzahn, 1997), WASA (Weske, 1996) and WIDE (Casati et al., 1996) 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. Fent et al. (2002), van der Aalst (2001) and Rinderle (2003) propose methods to migrate instances from the old process definition to the new one without introducing errors.

The third approach consists in 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. This is one of the branches that are followed by the COO project (Godart et al., 1999) and by other similar work (Georgakopoulos, 1999; Schuster et al., 2000). In Mobile (Jablonski, 1994), the authors define several perspectives (functional, behavioural, informational, organisational and 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 Georgakopoulos (1999), Godart et al. (1999) and Schuster et al. (2000) other examples of descriptive modelling are presented as techniques for compact modelling. The authors propose simple modelling constructs that better represent real and complex work patterns to be used, instead of a composition of elementary constructs.

The first two approaches consider flexibility at the level of the process execution 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 first approach considers the model as a guide to reach a goal and thus, activities are not controlled; all the advantages of explicit coordination are lost. The second approach allows to dynamically modify the process model by defining operators for changing the order of activities and inserting new activities. Supporting dynamic modifications is an important requirement for cooperative applications. However, contrary to presented models, a posteriori flexibility (by dynamic modifications) must be combined with a priori flexibility (allowing anticipation and intermediate results exchange). The third approach defines new modelling constructs for representing different work patterns which would be difficult or impossible to model in classical models. Cooperative processes being highly dynamic and unpredictable, it is difficult to model and to limit users to a given set of patterns.

In this paper, we consider a fourth way, which is not based on the way the process model is used or instantiated, neither on the way it can be evolved or modelled, but which adds flexibility in the workflow management system execution engine itself. This has the advantage of retaining the simplicity of the classical model and may also be adapted to other approaches. Our point of view is that the models that have been already proposed are most of the time very complex and not easy to grasp for designer and end user. Our proposal is a first step towards a simple model that could support a more flexible execution suited to engineering processes.

The exchange of intermediate results between activities is supported in some workflow models (Hagen and Alonso, 1999; Joeris, 1999).

In Hagen and Alonso (1999), an event-based communication mechanism allows the exchange of information between activities or subprocesses. Thus, activities in a subprocess that depend on the results of activities in another subprocess, can be started as soon as appropriate messages are received and do not have to wait until the whole subprocess is finished. While, here, the events are used to start an activity, we allow the exchange of information all the time during activities execution.

Joeris (1999) proposes a behaviour definition for a task as the basis for modelling less-restrictive workflows as well as supporting dynamic workflow changes. New control flow dependencies can be specified; in particular the simultaneous dependency allows controlling data exchange between simultaneous active tasks, assuring that the dependent task does not terminate before the preceding task. In contrast with our approach, the exchange of intermediate results is included in the model, though anticipated during process modelling step.

The goal of work presented in Kafeza and Karlapalem (2001) is also to accelerate the execution of workflow processes. However, their goal is to speed up the execution

of some workflow instances (those that are urgent for users), by increasing the priorities of their constituent tasks in the agent queues tasks. Two kind of algorithms for rescheduling activities in agent queues are presented and compared by experimental results. One-time speed-up algorithm tries to achieve as much speed up possible as soon as possible, whereas the second algorithm tries to achieve the required speed-up over time until the process instance completes its execution. If these algorithms can be used to speed up the execution of some process instances by slowing down instances less urgent, our goal is to accelerate the execution of each instance by improving the parallelism between activities.

Even if a lot of research work was dedicated to improving workflow flexibility, there is still not a generic solution allowing a flexible support for any kind of process. The flexibility of process systems is still an open problem. The existing approaches address different requirements and must not be considered as antagonistic. As an example, the operators allowing dynamic modifications could be applied to our model. More generally, we understood during different experiments in different application domains that these approaches are complementary and that better we would be able to make several of them cohabit, better we would provide support to codesign and coengineering processes. In fact, in a codesign and coengineering application, it does not exist one single process but several ones, and 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.

## 6 Conclusion

In this paper, we have presented a simple yet powerful way to modify workflow systems' classical behaviour to provide more flexible execution of processes. Compared with other approaches that try to provide flexibility in workflow models, the one we propose is simpler to understand for end users. This simplicity is essential in cooperative processes enabling graphical representation, helping their participants to situate in the context of the process and facilitating manual interventions for dynamic modifications. We showed that extending existing systems with anticipation is simple since it requires just modifying the state transition management of the execution engine and some adaptation to the way data are transmitted between activities. Anticipation also provides substantial benefits regarding activities execution, especially for interactive activities:

- parallelism of execution between successive activities
- possibility of early feedback between successive activities and
- potential acceleration of the overall execution.

Of course, there is also the risk of doing some extra work but as we mainly target interactive activities, we believe

that users are able to evaluate the opportunity to use anticipation and what they may gain from it. Furthermore, we plan to adapt the algorithms developed in Grigori et al. (2001) to use analysis of different executions of the same process with and without anticipation in order to help users to decide if anticipation should be used or not. This can be helpful in production processes where some paths in the workflow are taken most of the time given some initial conditions.

The next step of this work will be to consolidate the integration of the workflow model with cooperative transactions that we presented in Grigori et al. (2003). Having as our objective the simplicity of the workflow model, we will render the transactional support transparent to the user, by embedding cooperative transactions in a set of cooperation patterns, at the same level as MCI patterns (Schuster et al., 2000).

In the continuation of this study, an ongoing objective is to make usage analysis in realistic situations taking advantage of the applicative projects introduced above. We are also concerned with the integration of the COO-flow model with other workflow models, in the context of an intranet process or a multi-enterprise process.

## References

- Agostini, A. and De Michelis, G. (1996) 'Modeling the document flow within a cooperative process as a resource for action', *Technical Report CTL-DSI*, University of Milano.
- Bastos, R.M., Dubugras, D. and Ruiz, V. (2002) 'Extending UML activity diagrams for workflow modeling', *Production Systems, 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, Vol. 2, No. 10, Big Island, Hawaii.
- Bruno, G., Torchiano, M. and Agarwal, R. (2002) 'UML enterprise instance models', *Proceeding of International Conference on Information Technology*, Bhubaneswar, India December, 2002.
- Canals, G., et al. (1998) 'COO approach to support cooperation in software developments', *IEE Proceedings Software Engineering*, Vol. 145, Nos. 2-3, pp.79-84.
- Casati, F., et al. (1996) 'Workflow evolution', *Fifteenth International Conference On Conceptual Modeling (ER'96)*.
- Dayal, U., Hsu, M. and Ladin, R. (2001) 'Business process coordination: state of the art, trends and open issue', *Twenty Seventh International Conference on Very Large Data Bases (VLDB)*, Roma.
- Dunn, R.H. (1990) *Software Quality: Concepts and Plans*, Englewood Cliffs, NJ: Prentice Hall.
- Ellis, C. and Maltzahn, C. (1997) 'Chautauqua workflow system', *Thirtieth Hawaii International Conference On System Sciences*, Information System Track.
- Ellis, C.A., Keddara, K. and Rozenberg, G. (1995) 'Dynamic change within workflow systems', *Proceedings of ACM Conference on Organisational Computing Systems*, pp.10-22.
- Fent, A., Reiter, H. and Freitag, H. (2002) 'Design for change', *Evolving Workflow Specifications in ULTRAflow*, CAISE 2002, LNCS 2348, pp.516-534.
- Georgakopoulos, D. (1999) 'Collaboration process management for advanced applications', *International Process Technology Workshop*.

- Godart, C., Molli, P., Oster, G., Perrin, O., Skaf-Molli, H., Ray, P. and Rabhi, F. (2004) 'The toxicfarm integrated cooperation framework for virtual teams', *Distributed and Parallel Databases*, Vol. 15, No. 1, pp.67–88.
- Godart, C., Perrin, O. and Skaf, H.C. (1999) 'COO: a workflow operator to improve cooperation modeling in virtual processes', *Ninth International Workshop on Research Issues in Data Engineering Information Technology for Virtual Enterprises (RIDEVE'99)*.
- Grégory, N. (1999) 'Etude clinique d'une situation de conception de produit. Vers une pragmatique de la conception', PhD Thesis in Social Psychology, University of Nancy 2: Nancy, p.239.
- Grigori, D. (2001) 'Enhancing workflow flexibility for cooperative processes definition and enactment', PhD Thesis in Computer Sciences, University Henri Poincaré Nancy I, Nancy, p.135.
- Grigori, D., Casati, F., Dayal, U. and Shan, M.C. (2001) 'Improving business process quality through exception understanding, prediction and preventing, VLDB'2001', *Twenty Seventh International Conference on Very Large Data Bases*, Rome, pp.159–168.
- Grigori, D., Charoy, F. and Godart, C. (2001a) 'Anticipation to enhance flexibility of workflow execution', *DEXA'2001, 12th International Conference on Database and Expert Systems Applications*, Munich, LNCS 2113, pp.264–273.
- Grigori, D., Charoy, F. and Godart, C. (2001b) 'Flexible data management and execution to support cooperative workflow: the COO approach', *The Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'01)*, Beijing, China.
- Grigori, D., Charoy, F. and Godart, C. (2004) 'COO-flow: a process technology to support cooperative processes', *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Special Issue: Best Papers from SEKE 2003, Vol. 14, No. 1.
- Gruhn, V. (1995) 'Business process modeling and workflow management', *International Journal of Cooperative Information System*, Vol. 4, Nos. 2–3, pp.145–164.
- Hagen, C. and Alonso, G. (1999) 'Beyond the black box: event-based inter-process communication in process support systems', *Ninth International Conference on Distributed Computing Systems (ICDCS 99)*, Austin, TX.
- Harel, D. (1988) 'On visual formalisms', *Communications of the ACM*, Vol. 31, No. 5, pp.514–530.
- Hoare, C.A.R. (1985) *Communicating Sequential Processes*, Prentice-Hall International.
- Hollinsworth, D. (1995) *The Workflow Reference Model*, Technical Report TC00–1003, Workflow Management Coalition.
- Jablonski, S. (1994) 'Mobile: a modular workflow model and architecture', *Fourth International Working Conference on Dynamic Modeling and Information Systems*, Noordwijkerhout, NL.
- Joeris, G. (1999) 'Defining flexible workflow execution behaviors. In enterprise-wide and cross-enterprise workflow management – concepts, systems, applications', *GI Workshop Proceedings – Informatik'99*, Ulmer Informatik Berichte Nr. 99-07, University of Ulm.
- Kafeza, E. and Karlapalem, K. (2001) 'Speeding Up CapBasED-AMS activities through multi-agent scheduling', *Proceedings of the Second International Workshop on Multi-Agent-Based Simulation-Revised and Additional Papers*, Springer-Verlag, pp.119–132.
- Kazman, R., Bass, L., Abowd, G. and Webb, M. (1994) 'SAAM: a method for analyzing the properties of software architectures', *Proceedings of International Conference on Software Engineering – ICSE'16*, Sorrento, Italy, pp.81–90.
- Klein, M., Dellarocas, C. and Bernstein, A. (Eds) (2000) 'Computer supported cooperative work (CSCW)', *The Journal of Collaborative Computing*, Vol. 9, Kluwer, Special Issue on Adaptive Workflow Systems.
- Leymann, F. and Altenhuber, W. (1994) 'Managing business processes as an information resource', *IBM Systems Journal*, Vol. 33, No. 2.
- Leymann, F. and Roller, D. (1999) *Production Workflow*, Prentice Hall.
- Milner, R. (1980) *A calculus of Communicating Systems*, LNCS, Vol. 92, Springer Verlag, ISBN 0387102353.
- Nutt, G.J. (1996) 'The evolution toward flexible workflow systems', *Distributed Systems Engineering*, Vol. 3, No. 4, pp.276–294.
- Reichert, M. and Dadam, P. (1998) 'ADEPTflex – supporting dynamic changes of workflows without losing control', *Journal of Intelligent Information Systems*, Vol. 10.
- Rinderle, S., Reichert, M. and Dadam, P. (2003) 'Evaluation of correctness criteria for dynamic workflow changes', *Proceedings of International Conference on Business Process Management (BPM'03)*, Eindhoven, Netherlands June.
- Schuster, H., Baker, D., Cichocki, A., Georgakopoulos, D. and Rusinkiewicz, M. (2000) 'The collaboration management infrastructure', *Sixteenth International Conference on Data Engineering*, 28 February–03 March, San Diego, CA.
- Suchmann, L.A. (1987) *Plans and Situated Action. The Problem of Human–Machine Communication*, Cambridge: University Press.
- van der Aalst, W.M.P. (1998) 'The application of Petri nets to workflow management', *The Journal of Circuits, Systems and Computers*, Vol. 8, No. 1, pp.21–66.
- van der Aalst, W.M.P. (2001) 'Exterminating the dynamic change bug. A concrete approach to support workflow change', *Information Systems Frontiers*, Vol. 3, No. 3, pp.297–317.
- Weske, M. (1996) 'Flexible modeling and execution of workflow activities', *Thirty-one Hawaii International Conference on System Sciences*, Software Technology Track, Vol. 7.
- Weske, M. and Vossen, G. (1998) 'Workflow languages', in P. Bernus, K. Mertins and G. Schmidt (Eds) *Handbook on Architectures of Information Systems. (International Handbooks on Information Systems)*, Berlin: Springer, pp.359–379.
- Wirtz, G., Weske, M. and Giese, H. (2001) 'The OCoN approach to workflow modeling in object-oriented systems', in E. Stohr and L. Zhao (Eds). *Information Systems Frontiers*, Kluwer Academic Publishers, Vol. 3, No. 3, pp.357–376.
- Wodtke, D., Weißenfels, J., Weikum, G. and Kotz, D.A. (1996) 'The mentor project: steps toward enterprise-wide workflow management', *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, LA, pp.556–565.
- Zhang, A., Nodine, M., Bhargava, B. and Bukhres, O. (1994) 'Ensuring relaxed atomicity for flexible transactions in multidatabase systems', *Proceedings of ACM-SIGMOD International Conference on Management of Data*.