Cooperative Workflows to Coordinate Asynchronous Cooperative Applications in a Simple Way

Claude Godart, François Charoy, Olivier Perrin, Hala Skaf

Cooperative Workflows to Coordinate Asynchronous Cooperative Applications in a Simple Way.

C. Godart, F. Charoy, O.Perrin and H. Skaf-Molli
LORIA-INRIA
Campus Scientifique, BP 239,
54506 Vandœuvre-lès-Nancy Cedex - FRANCE
E-mail: {godart}@loria.fr

Abstract—
Current workflow models are mainly concerned with the automation of administrative and production business processes. These processes coordinate well defined activities which execute in isolation, i.e. synchronize only at their start/terminate states. If these models apply efficiently for a class of applications, they show their limits when one wants to model the subtlety of cooperative interactions as they occur in more creative processes, typically co-design and co-engineering processes.

In this paper, we introduce the concept of *cooperative workflow*, i.e. a workflow model which extends classical workflow models with capabilities to synchronize activities interacting not only when they start and they terminate, but also at any point of their execution. In the spirit of the workflow approach, modeling and enactment of cooperative workflows (must) remain simple.

Keywords—**Cooperation, Workflow, Transaction, Virtual enterprise**

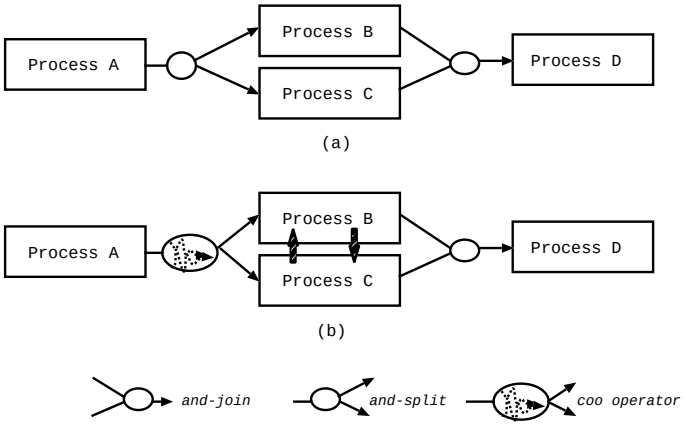

Fig. 1. From classical to cooperative workflows

I. Introduction

Current workflow models (we take [1] as a reference for definitions) are mainly concerned with the automation of administrative and production business processes. These processes coordinate well defined activities which execute in isolation, i.e. synchronize only at their start/terminate states. If these models apply efficiently for a class of applications, they show their limits when one wants to model the subtlety of cooperative interactions as they occur in more creative processes, typically co-design and co-engineering processes.

The objective of this paper is to deepen the concept of *cooperative workflow* we shortly introduced during RIDE'99 [2], [3]. The idea is to extend classical workflow models with capabilities to synchronize activities interacting not only when they start and they terminate, but also at any point of their executions (cf. figure 1).

However, in the spirit of the workflow approach, modeling and enacting cooperative workflows must remain simple. To support this requirement, we promote a transactional approach and a new operator to coordinate cooperating activities. This operator combines with traditional operators[1] as defined in [4].

Section II gives more motivations for our proposition. Section III gives a semantics to the concept of cooperative workflow by deepening the COO transaction model and introducing the *coo* operator. Section IV discusses the feasibility of our approach by relating some experiences. In section V, we analyse the advantages and the drawbacks of our model. Finally, section VI concludes.

[1]- *and-split* operator: when Process A terminates, it is followed by Process B and Process C which execute in parallel (in isolation)
- *and-join* operator : when Process B and Process C terminate, they are followed by process D
- *coo* operator : when Process A terminates, it is followed by Process B and Process C which execute in parallel but not in isolation; it is forecasted that they can (must) exchange data when executing

II. Cooperative workflow – Intuitive definition

A. A Motivating example

In this section, we develop an example of co-engineering taken in the domain of building construction to illustrate the kinds of interactions we are interested in and for which current workflow models show some limits.

Suppose a building trade application in which three partners (an architect, a reaserch engineer and a building contractor) cooperate. The architect has the responsibility of producing plans responding to a set of requirements given by a client. Based on the plans of the architect and on its proper expertise, the research engineer makes the main technological choices. The third partner, the building contractor has the responsibility to manufacture the wood components and finally to assemble them on building site.

Suppose a scenario in which the engineer chooses a wood technology. An iterative view of this process is depicted in figure 2(a). But real processes do not execute in this way. Processes corresponding to the three partners are more intricated and execute rather in parallel (it was the case in our case study) than in isolation: they exchange documents when executing with the objective to have a rapid feedback and to point out risks in the construction as soon as possible. In one scenario of our case study, in which the engineer choose a wood technology, due to transportation constraints (size of wood wall components), the building contractor asked the engineer to move an opening, and this decision had a lot of impacts especially with regards to the client wishes. Introduction of a model as in 2(a) can delay the evidence of this problem and contribute to a poor acceptance of the workflow system. A better way to do things is to allow the architect and the engineer on the one hand, the engineer and the contractor on the other hand, to cooperate by early exchanging documents (intermediate results) when executing. That is what figure 2(b) illustrates (see section III-B.2).

More motivating examples can be found in [3].

B. A set of representative cooperation patterns ?

We can extract the following characteristics from the process depicted above :

1. activities interact by exchanging artefacts, generally documents, which represent concepts concretizing the ideas of human agents,
2. in most cases, an agent which distributes an artefact waits for a feedback from cooperating agents : there is some goodwill to cooperate,
3. in most cases, processes are iterative and each artefact exists in several cooperative versions. Globally, the quality, in terms of requirement satisfaction or goal achievement, of the series of versions increases and converges towards a common agreement between the agents,
4. in most cases, artefacts are related each other and build a network of concepts. This network converges towards an agreement which satisfies the local agreement on each artefact.

To generalize this behavior, we have pointed out some intuitive patterns of cooperation. The three following are the more characteristic :

- *producer/consumer.* Two activities follow a *producer/consumer* pattern if one has the responsibility to modify an artefact and the other reads this artefact to integrate it in its proper work. Producers and consumers can momentarily see different versions of the artefact, but the consumer must see the final version stabilized by the producer.
- *developer/inspector.* Two activities follow a *developer/inspector* pattern if one (the developer) produces an artefact and the second (the inspector) reads one or several successive versions of this artefact with the objective to review it. Reviews become also shared artefacts.
- *cooperative write.* Two activities follow a *cooperative write* pattern if they develop two complementary versions of the same artefact and must merge their modifications before to conclude. In addition, they can make visible some intermediate version of their work before to conclude.

We have experimented that these patterns and their combinations allow to model a large number of situations. Nevertheless, we are deepening this classification [5].

C. Cooperative workflow : a definition

A cooperative workflow is a workflow where some activities executing in parallel can share some intermediate results during execution.

Here there are two main questions: (1) can activities in traditional workflow share intermediate results when executing ? (2) can the behavior depicted in section II-A be modeled with isolated activities ?

The first question was largely debated, as example during the International Process Technology Workshop [6]. Our synthetic point of view is that an activity in a traditional workflow is a black box, with inputs and outputs, but with no intermediate results visible. This is in accordance with WfMC definitions [1] and with most workflow specialists' points of view, even if some industrial products allow some exception to this

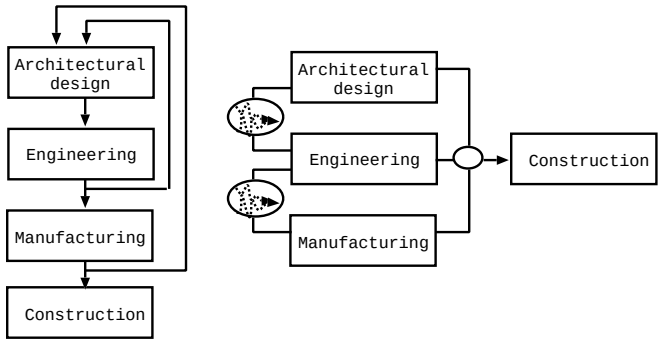

Fig. 2. Building trade example

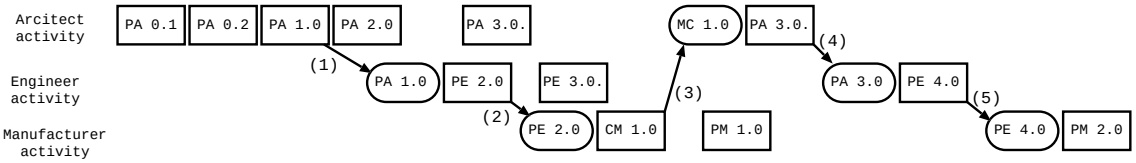

Fig. 3. A cooperative execution

rule (but in this case, semantics of parallel executions become quite unclear).

Concerning the second question, let us consider a possible execution of the building construction example (cf. II-A) depicted in figure 3. It tracks the exchange of results between the three main roles: architect, engineer and manufacturer. This based on 4 objects: the architect plan (PA), the engineer plan (PE), the manufacturer plan (PM) and some comments by the manufacturer (MC). In interaction (1), the architect provides a first version of his plan to allow the engineer to start his work quickly and/or to get a feedback on a first version. The objective of interaction (2) is the same, but between the engineer and the architect and with the engineer plan. Interaction (3) is different; it reports on a problem : typically, the need to move an opening due to transportation constraints. Interaction (4) responds to the need of modification, it is taken into account by the engineer which modifies its plan and transfers it to the manufacturer.

Clearly, such an execution cannot be (easily) modeled by traditional workflow models. Regarding traditional workflow operators, it can be deduced from interaction (1) that the architect activity precedes the engineer activity, from interaction (2) that the engineer activity precedes the manufacturer activity. Troubles arrive when we consider interaction (3) which introduces a first design iteration and breaks the sequence of things. One can try to model it by predefined iterations of sequences of activities, but the problem is that the iteration points cannot be forecasted.

As a conclusion, modelling this kind of behavior with traditional workflow can be only vague and not efficient. However, we think that there is a place for the concept of cooperative workflow which would provide support for such processes while providing a simple language to model these processes, keeping the workflow philosophy.

III. Cooperative workflows : Semantics

In this section, we make a particular instanciation of the cooperative workflow concept, based on the idea of COO-transaction [7]. In a first time, we present this model. In a second time, we show how COO transactions responds to our objective. In a third time, we explain how the solution supports scalibility.

A. COO transactions

The main reason for which traditional workflow does not fit design activities is that these activities can synchronize only when starting and terminating. Activities in traditional workflow execute like isolated transactions, even if atomicity is not systematically required (*terminated* state vs. *completed* state). At the opposite, we are interested in creative activities

which interact when executing at non forecasted execution points. Generally, these interactions are based on document exchanges. To provide workflow with the capability to interact when executing, one way is to allow activities to execute as open transactions. The COO transaction model is an open transaction model dedicated to support intercations, cooperation between creative activities.

As the objective of this paper is not to deepen this model of transactions, we content us here with the main characteristics of this model, starting with a synchronization protocol which asserts correctness of interactions (corresponding to *concurrency atomicity* in traditional transactions [8]) as defined by COO-serializability. Deepening this criteria is not necessary for the understanding of the concept of cooperative workflow. For more about COO-transaction, COO-serializability and related work see [7].

A.1 Protocol

We depict here the COO protocol which provides an operational semantics of this model by asserting the correctness of interactions in executions it accepts:

1. a result produced before the end of a process is always an intermediate result. Users can call, at any time, the operation $IR - write$ to produce an intermediate result.
2. a result produced at the end of a process is a final result. All final results are produced atomically during the execution of the *terminate* operation.
3. a process that produces an intermediate result must produce a corresponding final result. The protocol collects all the objects that were "$IR - written$" by the process and automatically produces a final result for each of them during the termination phase of the process.
4. if a process reads an intermediate result, then it must read the corresponding final result. The system maintains a *dependency relationship* between processes to memorize the fact that a process reads an intermediate result of another one. When an activity A_1 reads the intermediate value of an object x produced by a process A_0, then a dependency $A_1 \longrightarrow A_0$ is created. When the process A_1 reads a value of x and A_0 is terminated (i.e. when it has produced its final result), the dependency is removed.
5. a process cannot terminate if it is still dependent on another one. If a process tries to terminate without reading the final value of some object after a previous access to an intermediate value of this object, the *terminate* operation is aborted and the process remains active.
6. processes involved in a cyclic dependency graph form a *group* of processes.
7. a group-member process can start a *group termination* by trying to terminate itself. The *terminate* operation in this case produces a set of potentially final results and changes the state of the process from *active* to *ready to terminate* (RTT).
8. when a group-member process tries to *terminate* and all the other group-members are in the RTT state, then all the processes are terminated simultaneously. Potentially final results are definitely promoted to final results.
9. when a group-member process tries to *terminate* and there is another group-member still *active*, then it produces new potentially final results and enters the RTT state.
10. if a group member produces a new intermediate result during the group termination phase, then this termination tentative is aborted, and all the group-members re-enter the *active* state. This is the way for an activity to clearly indicate its disagreement with the object values produced by the group, and to ask for more work on the shared objects.

One can observe that, as an activity is automatically added to a cooperative group upon its participation in a cycle, we risk having all activities ending up in one huge group which may in return prevent the group to converge at all. In fact, if this phenomenon cannot be theoretically excluded, it is limited as the result of the natural flow of activities which globally ensures that applications progress towards their goal.

A.2 Consistency

One can also notice that, if the protocol imposes a process, which has read an intermediate result x, to read the final version of x, it imposes nothing to prevent some inconsistencies to appear: if the value of x is used to modify another object y, the protocol do not impose y to be updated with the new value of x.

We consider that it is a problem of internal consistency of each activity and we completely delegate the propagation of final version reads to each activity. This allows a clear distinction between the correctness of interactions between activities, and the individual correctness of activities, as in traditional transaction models. We show in [9] the impact of intermediate results on consistency and how this characteristic can be managed locally.

B. Modeling cooperative workflows with COO-transactions

In this section, we illustrate how COO-transactions can provide active support for safe cooperative work-

flows. For this purpose, let us introduce an operator, to be compared (and combined) with those of [4]. In a first time, we start with a general form. Then we show how this general form can be restricted. Especially, we explain how traditional transactions are a special case of COO-transactions.

B.1 The *coo* operator – By default form

Notations $coo(p_1, p_2)$ To simplify, let us start with a binary operator. This means that process p_1 can see intermediate results of process p_2, and mutually that process p_2 can see intermediate results of process p_1 without restriction.

Figure 2(b) describes the *building trade* example using the *coo* operator. The parallel processes cooperate in the sense of the *coo* operator before to converge in a common thread of control (*and-join* of [4]) where the construction can start.

The semantics of *coo* in its default form is exactly this defined by the COO protocol and specified by the COO-serilalizability (cf. III).

Properties In this form, the $coo(p_1, p_2)$ is symmetrical : $coo(p_1, p_2)$ implies $coo(p_2, p_1)$. coo(*architectural design*, *engineering*) implies coo(*engineering*, *architectural design*). It is also transitive : $coo(p_1, p_2)$ and $coo(p_2, p_3)$ implies $coo(p_1, p_3)$. In figure 2(b), we have explicitly, coo(*architectural design*, *engineering*) and coo(*engineering*, *manufacturing*). This implies also coo(*architectural design*, *manufacturing*). These properties apply for the by default form of *coo*. We discuss later the configurable form.

Collaboration with other operators *coo* does not enforce any restriction on the structure of the process it synchronizes. It can be combined with other operators depending on the logic of the application and one operand p_1 and p_2 can be refined as a cooperative workflow.

It is interesting to notice here the flexibility of the approach. In fact, using the *coo* operator, we do not define in a rigid way how processes will interact but we simply track dependencies between processes and finally impose convergence on the value of final results. As a consequence, the same process model can be instantiated in different ways provided processes converge towards common consistent final results. This allows to manage in the same way a family of processes which would request to manage deviation [10] in a traditional approach.

B.2 The *coo* operator – Configuration

The *coo* operator as introduced above does not imply any constraint neither on the objects which can be shared nor on the way they are shared. That is the more simple form to understand and to use and we consider it is the form by default. However, it is clear that it must be possible to set some restrictions on the interactions between the participants in a cooperation.

To support this, the *coo* operator can be configured (or parameterized). The syntax of a cooperation between two processes has the following general form :

$$coo(p_1, p_2)(o_i, direction, mode)^*$$

where o_i is an object and $*$ has the current meaning of *zero or several times* and where *direction* and *mode* have the following meaning :

- *direction* can take three values ($\longleftrightarrow$, $\longrightarrow$, $\longleftarrow$, $\not\longleftrightarrow$) and indicates, respectively, if the interaction is bidirectional, or from p_1 to p_2, or from p_2 to p_1, or is forbidden,
- *mode* can take two values (*all* or *atdemand*) and indicates if all intermediate results must be consumed (*all*) or not (*atdemand*).

More precisely, concerning *direction* :

- $\longrightarrow$ means also that p_1 can put some successive intermediate results of o_i at p_2 disposal. p_2 can only read o_i intermediate results.
- $\longleftarrow$ means also that p_2 can put some successive intermediate results of o_i at p_1 disposal. p_1 can only read o_i intermediate results.
- $\longleftrightarrow$ means also that both p_1 and p_2 can modify at the same time the same object and can put some successive intermediate results of o_i at the other process disposal.
- $\not\longleftrightarrow$ means that the object o_i cannot be shared before the process, which modifies it, terminates.

direction and *mode* can be considered as constraints, added to the form by default, which complement other constraints like pre- and post-conditions.

The form by default depicts the case where objets are specified as shared $\longleftrightarrow$ *direction* and *atdemand* mode. It is the more free way of cooperation.

Back to *coo* semantics Above, we gave the semantics of the unconstrained *coo* operator, i.e. a cooperation in which all objets are shared by default, in $\longleftrightarrow$ *direction* and *atdemand* mode. Configuration (or parameterization) simply restricts this semantics by adding constrains on interactions : the set of correct executions is a subset of the correct executions specified by the COO-serializability introduced above. This restriction can be easily modeled. If interactions are oriented, the consuming process can only read intermediate result(s) of the other; if all intermediate results must be consumed, a process must read all intermediate results in a FIFO mode. This can be simply expressed and implemented.

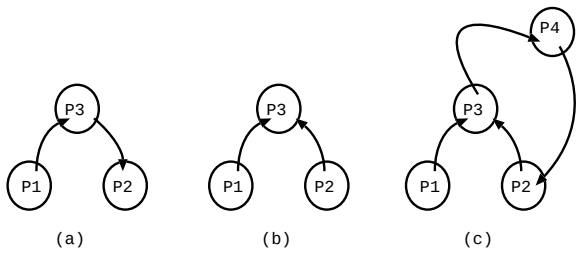

Fig. 4. Relationships between concurrent and cooperative processes

Back to properties of the *coo* operator $\longrightarrow$ and $\longleftarrow$ break the property of symmetry because of the unbalanced role of p_1 and p_2 in these cases. The notion of transitivity becomes also unclear. How to interpret the combination of $coo(p_1, p_2)(o_i, \longleftrightarrow, atdemand)$ with $coo(p_2, p_3)(o_i, \longrightarrow, atdemand)$, does it really imply $coo(p_1, p_3)(o_i, \longrightarrow, atdemand)$? What is exactly the role of p_2 in this configuration ? How to interpret the combination of $coo(p_1, p_2)(o_i, \longrightarrow, atdemand)$ with $coo(p_2, p_3)(o_i, \longleftarrow, atdemand)$. Is this possible if $coo(p_1, p_2)(o_i, \longleftrightarrow, atdemand)$ is not specified somewhere ? More generally, it is clear that configuration introduces new problems which are hardly related to discretionary access right control. We still have to deepen this aspect of the model.

One can consider configuration as too much difficult with regards to the target applications. In fact, it is not the case: such a configuration is not more difficult than to define access rights in data oriented application. In addition, configuration can be abstracted by some kinds of cooperation patterns, as introduced in section II-B [5].

B.3 Concurrent versus cooperative processes

It is important to situate our cooperative processes with regards to concurrent processes as used in traditional workflows. In fact, concurrency is a special case of cooperation: two concurrent processes are two processes which do not exchange intermediate results when executing, but only when they start and terminate. In this case, *mode* has no meaning.

$$concurrent(p_1, p_2) \equiv \forall i, coo(p_1, p_2)(o_i, \not\longleftrightarrow, ?)$$

This is formally well founded due to the fact that a COO-serializable execution, in which no intermediate result produced by a process is read by another process, is a serializable execution [7].

Establishing this relationship between concurrent and cooperative processes is especially interesting, because it allows to manage in the same process concurrent and cooperative sub-processes.

However, this imposes also some precautions. Let us consider figure 4 with the hypothesis $concurrent(p_1, p_2)$. Figure 4 (a) illustrates the fact that p_1 and p_2 cannot cooperate with a third process p_3 following cooperations $coo(p_1, p_3)(o_i, \longrightarrow, atdemand)$ and $coo(p_3, p_2)(o_i, \longrightarrow, atdemand)$: p_2 sees indirectly an intermediate result of p_1. However, in some cases, p_1 and p_2 can cooperate with a common process p_3 as illustrated in figure 4 (b). But considering directly connected processes is not sufficient to assert isolation between p_1 and p_2) : isolation can be broken transitively (cf. figure 4 (c)).

Deepening the rules of cooperation in presence of concurrent processes is out of the scope of this paper. In its general nature, this problem is indecidable (it is as complex as proving view serializability). However, restrictive protocols can be developed to support isolation of two processes. One is to prevent a concurrent process to produce intermediate results. This seems not a too restrictive constraint with regards to the traditional idea of concurrency.

Maintaining or not a *concurrency* operator ?

Even if concurrency is a special case of cooperation, we propose to maintain an operator *concurrency* (parallel operator of [4]) for two reasons:

- some processes do not need cooperation,
- concurrency can be considered as a well known cooperation pattern: cooperation without exchange of intermediate results.

IV. Feasability

We have implemented the *coo* operator in several contexts.

In a first experience, we developed a software engineering environment, called *COO*, on top of a centralized object management system (compliant with the PCTE standard) [11]). If the object base was centralized (cf. figure 5 (a)), access points were distributed through a network of Unix based workstations. Correctness was based on COO-serializability.

The objective of the second experience was to take the *COO* environment from a centralized object base to a set of object bases, one object base per partner of a virtual enterprise (cf. figure5 (b)). It demonstrates the applicability of the protocol developed in the previous experience with very light modifications : COO-serializability is adapted to be assessable only with local logs of events (DisCOO-serializability) and the protocol is very close to the initial one, the main difference being in the choice of a coordinator to complete a group of transactions. The implementation is

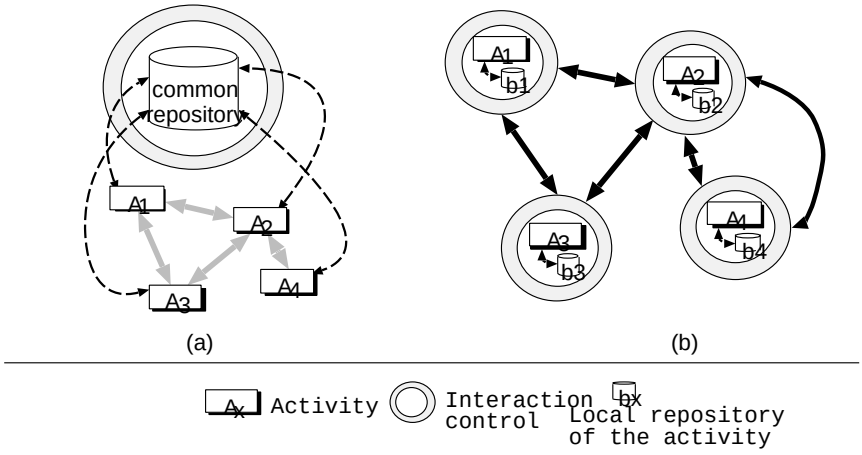

Fig. 5. Centralized and distributed architecture to cooperate

based on a CORBA bus [12].

We are now porting this experience on *Internet*. The principle is to develop a *virtual memory* for *virtual enterprise* [13]. This memory makes appear file systems (typically Windows files) of the partners as one entity. To allow locally the evaluation of predicates, some information is duplicated on different sites. To maintain the consistency of duplicates, updates are ordered thanks to an atomic broadcast. This memory makes the implementation of the protocol not so different in principles than in a centralized architecture. In addition, it provides support for disconnected work, as in most applications, a partner cannot remain connected to the network all the time. This work is under development.

V. Advantages and Drawbacks of our Approach — Related Work.

The main advance of cooperative workflow with regards to traditional workflow is its flexibility which allows to model creative processes more closely to reality than traditional workflow can. Cooperative workflow should provide a better degree of acceptance than traditional workflow for the automation of creative processes.

Our approach inherits also :

- the advantages of the workflow approach, especially the simplicity of modeling and automation,
- the advantages of the transactional approach, especially its solid foundations.

Concerning this model of transactions, it can be compared to other transaction models designed to support cooperation. We think that the advantage of our approach is that cooperation is, let say *native*, i.e. is the natural way to execute processes. In other words, the process of supporting iterative design is implicit and does not need to be declared. Taking the unified theory of concurrency as reference [14], [15], our approach is especially efficient for *retriable* activities, i.e. activities which can be replayed several times until they commit. The difference is that in our approach, iteration of an activity is a natural way of work: if iteration n completes, iterations $i, i < n$ are not aborted transactions, with the idea of tansactions which have no usefulness, but are simply useful step towards the final result. In other words, an "abort" of an iteration is not a setback, but a chance of interaction and feedback.

The main drawback is that it can be difficult to assume the convergence of the cooperating activities before they complete. In fact, the flexibility of the approach rests on the principle that activities can temporarily have non synchronized views of the shared objects, can view different versions of these objects. This can lead to an important divergence between activities which risk to have an important work to do before to be able to converge and complete. This is especially crucial if we integrate the fact that in most cases, convergence is forced due to time constraints (deadline). However, we can notice that this behavior is intensively used in practice, and that our approach which tracks dependencies between artefacts helps to the reconciliation of the different versions. In addition, we are working on divergence measurement and control, in the idea of [16].

Another drawback is the fact that the approach rests on the goodwill of organizations which must accept to make visible some intermediate results, i.e. results which are not products, but which can be considered as ideas, plans, and which can give information on

the patrimony of the organization. Any organization is not ready to make visible such results to any other organization. And in addition, an organization which shares such an object with another organization wants this enterprise to respect some rules of confidentiality. These are real problems which need to be deepened.

Regarding the approach developed in [17], it has the advantage to be better formalized and transparent to users. Its disavantage is to be less general, to respond only to a particular need of of flexibility, for conceptual applications like co-design and co-engineering It is not (directly) concerned with advanced applications like service provisioning in virtual enterprises, crisis mitigation [18], [19] .... Note also that our approach is compatible with this developed in [17].

VI. Conclusion

As we motivate in this paper, we think that cooperative workflows are feasible and have a lot of applications, typically, in the context of cooperative applicatins in virtual enterprises [2]. We have made several exprimentations in this direction (cf. IV), but probably that a broader application of the concept should get some work of normalization, as done for tradtionnal transactions. The approach has several drawbacks, and especially some drawbacks related to organizations, but we can anticipate with [20] that process modelling will become more and more crucial for the competitivity of organizations. Our short and midle term objectives are: to validate our approach in the context of our industrial projects, to study how our approach can help coordination in the context of virtual enterprises (cross-organizational workflows), to study the relatinships between coordination and direct communication: how coordinated activities provoke direct communication, how direct communication impact coordination (re-organization ...).

References

[1] Workflow Management Coalition, "The Workflow Reference Model," Tech. Rep. WFMC-TC-1003 Version 1.1, Workflow Management Coalition, january 1995.

[2] D. Georgakopoulos and L. Maciaszek, *Proceedings of 9th International Workshop on Research Issues in Data Engineering (Virtual Enterprise)*, IEEE Press, 1999.

[3] C. Godart et O. Perrin et H. Skaf, "COO: a workflow operator to improve cooperation modeling in virtual enterprises," in *9th IEEE RIDE, International Workshop on Research Issues in Data Engineering, Sydney*, March 1999.

[4] Workflow Management Coalition, "Terminology & Glossary," Tech. Rep. WFMC-TC-1011, Issue 2.0, Workflow Management Coalition, June 1997.

[5] S. Tata, G. Canals, and C. Godart, "Specifying interactions in Cooperative applications," in *Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99)*, Kaiserslautern , Germany, 1999.

[6] *International Process Technology Workshop*, http://www-adele.imag.fr/IPTW/, 1999.

[7] G. Canals, C. Godart, P. Molli, and M. Munier, "A Criterion to Enforce Correctness of Indirectly Cooperating Applications," *Information Sciences*, vol. 110/3-4, pp. 279–302, September 1998.

[8] P.A. Bernstein and E. Newcomer, *Principles of Transaction Processing*, Morgan Kaufmann, 1997.

[9] Hala Skaf, François Charoy, and Claude Godart, "Maintaining Shared Workspaces Consistency during Software Development," *Journal of Software Engineering and Knowledge Engineering*, vol. 9, no. 5, 1999.

[10] J. Herbst and D. Karagiannis, "Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models," in *Proceedings of the Ninth IEEE Conference on Database and Expert Systems Applications*, 1998.

[11] C. Godart, G. Canals, F. Charoy, P. Molli, and H. Skaf, "Designing and Implementing *COO*: Design Process, Architectural Style, Lessons Learned," in *International Conference on Software Engineering (ICSE18)*, 1996, IEEE Press.

[12] M. Munier, K. Benali, and C. Godart, "A transactional Approach for Cross-Organizational Cooperation," in *GlobeComm99 (Enterprise Applications and Services Symposium)*, Rio de Janeiro, Brazil, december 1999.

[13] G. Canals, P. Molli, and C. Godart, "Support for end user participation using replicated versions and group communication," *ACM SIGGROUP Bulletin*, 1999, To appear.

[14] H-J. Schek, G. Weikum, and H. Ye, "Towards a unifying theory of Concurrency, Control and Recovery," in *ACM Symposium on Principles of Database Systems (PODS)*, 1993, pp. 300–311.

[15] H. Schuldt, G. Alonso, and H-J. Schek, "Concurrency control and recovery in transactional process management," in *ACM Symposium on Principle of Database Systems (PODS'99)*, May 1999, pp. 1–11.

[16] K.L. Wu, P.S. Yu, and C. Pu, "Divergence control for espsilon-serialisability," in *Proceedings of the 8th International Conference on Data Engineering*, Phoenix, February 1992.

[17] C. Hagen and G. Alonso, "Beyond the Black Box : Event-based Inter-Process Communication in Process Suport Systems," in *19th ICDCS, International Conference on Distributed and Computing Systems*, 1999.

[18] D. Georgakopoulo, Hans Schuster, and M. Rusinkiewicz, "Collaboration Process Management for Advanced Application," in *[6]*, 1999.

[19] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke, "A Comprehensive Approach to Flexibility in Workflow Management Systems," in *WACC99, Work Activities Coordination and Collaboration, ACM Press*, San Francisco, USA, february 1999.

[20] T.W. Malone and al., "Tools for inventing organizations: toward a handbook of organizational processes," rapport technique http://ccs.mit.edu/ccswp198, Center for Coordination Sciences, MIT, 1988.