CORVETTE: a cooperative workflow for virtual teams coordination

Karim Baïna, François Charoy, Claude Godart, Daniela Grigori, Saad El Hadri, Hala Skaf, S. Akifuji, Toshiaki Sakaguchi, Yoko Seki, Masaichiro Yoshioka

CORVETTE: a cooperative workflow for virtual teams coordination

K. Baïna, F. Charoy, C. Godart*, D. Grigori, S. el Hadri and H. Skaf

ECOO Team, LORIA/INRIA, Campus Scientifique, BP 239,
54506 Vandœuvre Cedex, France
E-mail: Claude.Godart@loria.fr
http://www.loria.fr/equipes/ecoo
*Corresponding author

S. Akifuji, T. Sakaguchi, Y. Seki and M. Yoshioka

Hitachi SDL, Kawasaki Lab., 1099, Ohzenji, Asao-ku,
Kawasaki-shi, Kanagawa-ken, 215-0013, Japan
http://www.sdl.hitachi.co.jp/

Abstract: Workflow management systems are now widely deployed for handling administrative and production applications. In order that the workflow management systems keep supporting a larger range of applications, several research works have been launched to improve the workflow technology. This paper reports on the CORVETTE [1] project, an experiment in developing a cooperative workflow management system by integrating a workflow management system commercialised by Hitachi Ltd. with a cooperation technology proposed by INRIA. A cooperative workflow management system is a system gathering workflow management functionalities and having capabilities to manage cooperative behaviours characteristics of creative application processes (e.g. in codesign and coengineering). In this kind of applications, interactions between activities are characterised by being more subtle than in traditional applications (i.e. non-scheduled, unpredictable, dynamic, etc.). More precisely, CORVETTE was targeted to support coordination workflow processes of a virtual team working over the internet.

Keywords: workflow management systems; cooperative processes; coordination of virtual teams; cooperative transaction models.

Reference to this paper should be made as follows: Baïna, K., Charoy, F., Godart, C., Grigori, D., el Hadri, S., Skaf, H., Akifuji, S., Sakaguchi, T., Seki, Y. and Yoshioka, M. (2004) 'CORVETTE: a cooperative workflow development experiment', *Int. J. Networking and Virtual Organisations*, Vol. 2, No. 3, pp.232–245.

Biographical notes: Karim Baïna is an Assistant Professor at "Ecole Supérieure d'Informatique et d'Analyse des Systèmes" (ENSIAS), Rabat, Morocco. He obtained his PhD in Computer Sciences from University Henri Poincaré, Nancy 1, France, 2003. His research interests include cooperative systems, workflow management systems, group negotiation and decision systems.

François Charoy is an Assistant Professor of Information Systems at the University of Nancy 2. He received his 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.

Claude Godart is Full-Time Professor at University Henri Poincaré, Nancy 1, France. He is scientific leader of the ECOO project of INRIA interested in developing middleware for supporting cooperative work. His proper interests includes consistency of data in distributed systems, workflow models and systems, web services and virtual enterprises.

Daniela Grigori received her masters and PhD from the University Nancy I, France in 1998 and 2001, respectively. Since 2002 she is 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.

Hala Skaf-Molli is an Assistant Professor at the University Henri Poincaré, Nancy 1, France. She received her PhD in Computer Science from the same university in 1997. Her research interests include issues related consistency maintenance of the data mediating the cooperation between several partners, and combination of transactional techniques and transformational techniques for the synchronisation of copies.

Shunsuke Akifuji received his MS (Engineering) degree from University of Tokyo, Japan in 1988. He joined Hitachi, Ltd. in 1988 and is currently working for Hitachi (China) Investment Ltd., Research & Development Center. His research interests include issues related to workflow management system, enterprise applications and web services.

Toshiaki Sakaguchi received his MS (Science) degree from Kyoto University, Japan in 1990. He joined Hitachi, Ltd. in 1990, and is engaging in the development of a Collaboration Ware "BOXER" since 2001. His interests include issues related to workflow management system, collaboration software and web services. He has participated actively in the Workflow Management Facility standardisation in OMG.

Yoko Seki received her MS (Engineering) degree from Nagoya University, Japan in 1997 and is currently a Researcher at Hitachi, Ltd., Systems Development Laboratory. Her research interestsare in the areas of business process management, chiefly workflow management systems, and web services.

Masaichiro Yoshioka received his MS (Science) degree from Kyoto University, Japan in 1982. He joined Hitachi, Ltd. in 1982, and is engaging in Information and Communication Technology (ICT) business planning and R&D management since 2000. His business relates all the aspects of ICT areas and trends. He has established several research collaborations with research institutes including INRIA.

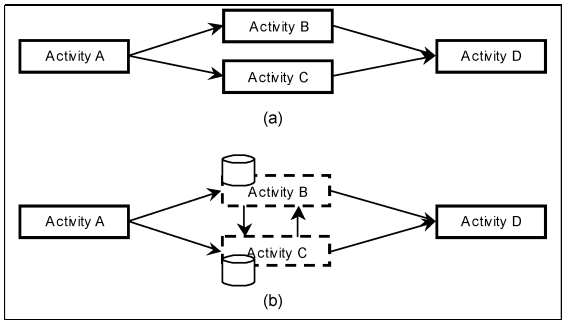
1 Introduction

Workflow management systems are now widely deployed for handling administrative and production applications. To address a larger range of applications, a lot of research has been launched to surpass the limits of current workflow technology [2–15]. Among them, some researches [4,5,16] concentrate on supporting creative processes, typically codesign and coengineering processes, where interactions between activities are more subtle than in traditional applications (i.e. non-scheduled, unpredictable, dynamic, etc.). We call a system that gathers workflow management functionalities and has capabilities to manage cooperative behaviours characteristic of creative applications as a cooperative workflow management system. This paper, written on the basis of [17], reports on the CORVETTE project. CORVETTE is an experiment in developing a cooperative workflow management system by integrating a workflow management system commercialised by Hitachi Ltd. with a cooperation technology developed by INRIA. More precisely, CORVETTE was targeted to support coordination processes of a virtual team working over the internet. It is organised as follows. Section 2 describes the motivation for cooperative workflow. Section 3 describes the CORVETTE system design. Section 4 sketches the implementation. Finally, Section 5 synthesises the experiment and discusses some conclusions.

2 Why cooperative workflow?

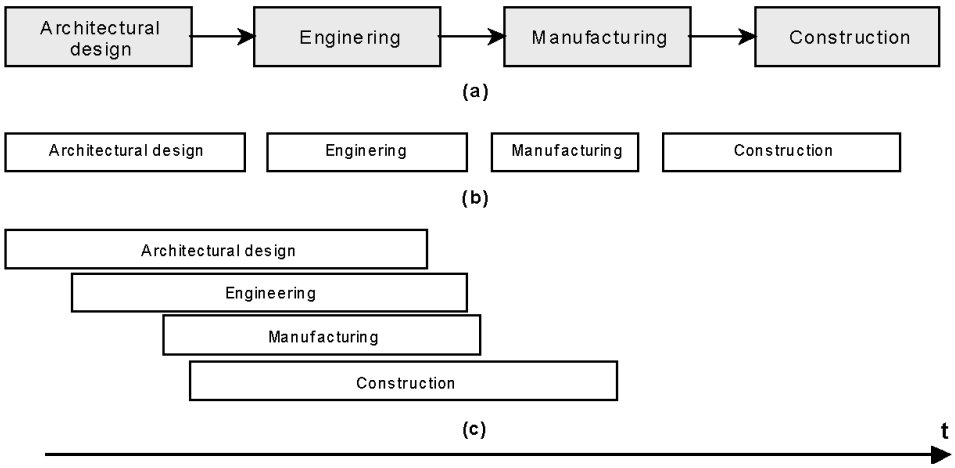
Current workflow models are mainly concerned with the automation of administrative and production processes. These processes coordinate well-defined activities which execute in isolation, i.e. synchronise only at their start/terminate states. Though these models work efficiently for a class of applications, their limitations become evident when one wants to model the subtlety of cooperative interactions as they occur in more creative processes, typically codesign and coengineering processes. Cooperative workflow (Figure 1) has been mainly introduced to overcome these limitations. A cooperative workflow is a workflow where some activities being executed in parallel can share some intermediate results during their execution [18,19]. A cooperative workflow has the capability to synchronise activities at any point of their executions and thus ensures the coordination between designers [19].

Figure 1 From classical to cooperative workflow (a) classical work flow: activities B and C are closed and execute in isolation and (b) collaborative workflow: activities B and C are porous and can share intermediate results when executing



To illustrate this, let us consider three partners cooperating within a building trade process (an architect, a research engineer and a building contractor). The architect has the responsibility of producing plans corresponding to a set of requirements given by a customer. Based on the plans of the architect and on his own expertise, the research engineer makes the main technological choices. The building contractor has the responsibility of manufacturing wood components and finally of assembling them on building site. This process is roughly described in Figure 2(a). A traditional (i.e. sequential) execution of this process is depicted in Figure 2(b). However, real processes do not execute really in this way. Processes corresponding to the three partners are more intricate and execute in parallel rather than in isolation. Actually, they exchange documents when executing with the objective to obtain a fast feedback and to point out risks in the construction as soon as possible. In general, definition of a model as in Figure 2(b) can delay a design problem and contribute to poor acceptance of the workflow management system on working sites. A better way to do things is to allow the architect, the engineer and the contractor to cooperate by exchanging intermediate results early (e.g. draft documents) when operating. This idea is illustrated in Figure 2(c) besides the fact that this organisation can dramatically decrease the total duration of a process.

Figure 2 Building trade example (a) building trade process model, (b) building trade serial execution and (c) building trade collaborative execution



Enabling interactions between parallel activities is a typical cooperation characteristic within creative applications that we are concerned with. One important question at this point is: can a traditional workflow management system (WFMS) model and enact such interactions efficiently? In more accepted definitions of a workflow management system [20–22], an activity is a black box with an input container that is filled before the activity starts its execution and an output container that is filled when the activity ends. Thus, in classical workflow management systems, visibility of intermediate results is categorically prohibited.

Therefore, such a WFMS cannot easily support such cooperative interactions. One can try to model it with iterations, knowing that the limits of this solution are easily reached. Interactions are difficult to forecast and their number increases exponentially with the number of activities [19]. Moreover, for practical opportunity, some products

allow activities to interact when executing, but in this case, semantics of parallel executions become quite unclear, or rather is delegated to the applications, under the responsibility of the programmer. As far as current systems go, they cannot easily model cooperative behaviours. There is, therefore an obvious need for improving workflow management systems with cooperative workflow technology.

3 CORVETTE design

The objective of the CORVETTE project is to examine the development of a cooperative workflow management system to coordinate a virtual team cooperating through a wide network, typically the internet. Cooperation has to be taken here in the sense of the ECOO team cooperation model that we will present below. The final system must be implemented by 'connecting' the *Hitachi WorkCoordinator* [23] workflow management system and the *ECOO MOTU cooperative system* [24], which provides complementary cooperation services, and especially COO [25] cooperative transactions. This achievement must consider the *WorkCoordinator* system as a black box and thus must not touch its source code. Thus, the implementation has to enrich the *WorkCoordinator* with new cooperation services that interact with workflow services only by invocating the *WorkCoordinator* CORBA application programmable interface or by monitoring its embedded Oracle database.

This section is organised in three steps. First, we introduce the contributing technologies, followed by presentation of the important design decisions, and finally, we illustrate the detailed architecture of CORVETTE.

3.1 Contributing technologies

We start with the ECOO cooperation technology represented by the *MOTU cooperative system* and then introduce the Hitachi workflow technology within the *WorkCoordinator* system.

3.1.1 ECOO cooperation technology: Motu system

The ECOO cooperation model is based on the ability for an activity to publish an intermediate result, as introduced in Section 2. More precisely, the ECOO cooperation model is characterised by four generic cooperation patterns as depicted below:

- *Producer/consumer*. Two activities follow a producer/consumer cooperation pattern; one has the responsibility to create/modify an object and the other reads this object to integrate it in its own work. Producers and consumers can momentarily see different versions of the same object, but the consumer must retrieve the final producer version.
- *Redactor/reviewer*. Two activities follow a redactor/reviewer cooperation pattern; the redactor creates/modifies an object and the reviewer reads one or several successive versions of this object with the objective of reviewing it. The corresponding review objects are also shared in their turn.

- *Cooperative write*. Two activities follow a cooperative write pattern; they develop a common object, work on two different versions and frequently merge their modifications. Finally, they have to agree on a common final version.
- *Concurrency.* Two activities follow a concurrency cooperation pattern if they must execute in isolation. In cooperative processes, some activities must be allowed to share some results, while other activities must be allowed to execute in isolation with respect to others.

Based on the ECOO cooperation model, the *MOTU cooperative system* [24] is a computer supported cooperative (or CSCW) tool on the lines of BSCW [26], TeamScope [27] or Sourceforge [28]. It proposes, among other cooperation services, workspace management, communication, coordination, group awareness [29] etc. The main *Motu* cooperation service components used in CORVETTE project are the workspace manager and the COO-transaction manager that have been streamlined to manage consistency of non-linear versioned document exchanges.

The COO-transaction protocol asserts *concurrency atomicity* of cooperative processes (i.e. correctness of interactions between processes exchanging results when executing). For this purpose, each process is encapsulated in a COO-transaction. Each COO-transaction executes in its private workspace and COO-transactions cooperate by transferring intermediate results between their workspaces. The COO-transaction protocol is a set of rules that sets constraints on these transfers. They are intuitively depicted in the following:

- A result produced before the end of a COO-transaction is always an intermediate result of this transaction. Users can produce an intermediate result (operation *IR-write*).
- We call as the final result, a result produced at the end of a COO-transaction. All final results are produced atomically during the execution of the *terminate* operation of the transaction. A COO-transaction that produces an intermediate result must produce a corresponding final result. The protocol collects all the objects that were *IR-written* by the activity and automatically produces a final result for each of them during the termination phase of the activity.
- If a transaction has read an intermediate result, then it must read the corresponding final result. The system maintains a *dependency relationship* between activities to memorise the fact that a transaction reads an intermediate result of another one. When a transaction A_1 reads the intermediate value of an object x produced by a transaction A_0, a dependency $A_1 \rightarrow A_0$ is created. When the transaction A_1 reads a value of x and A_0 is terminated (i.e. when it has produced its final result), the dependence is removed.
- A transaction cannot terminate if it is still dependent on another one. If a transaction 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 transaction remains active.
- Transactions involved in a cyclic dependence graph form a group of transactions.

- A group-member transaction 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 transaction from *active* to *ready to commit* (RTC).
- When a group-member transaction tries to terminate and all the other group-members are in the RTC state, then all the transactions are terminated simultaneously. Potentially final results are definitely promoted to final results.
- When a group-member transaction tries to terminate and another group-member is still active, then the former produces new potentially final results and enters the RTC state.
- 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 a transaction to clearly indicate its disagreement with the object values produced by the group, and to ask for more work on the shared object.

More formal definitions of COO-transaction model and relationships between concurrency, atomicity and consistency is given in [30,31].

3.1.2 Hitachi workflow technology: WorkCoordinator system

Based on the Work*f*low Management Coalition (or W*f*MC) workflow specifications [20], the *WorkCoordinator* workflow management system (or *WCO*) considers a workflow entity as a control flow graph with *activities* as nodes and inter-activities *transition*s as edges (i.e. and-join, and-split, or-join, or-split, sequence, etc.). An activity has a description (textual field), a deadline (in days), and a post condition (an SQL statement) which defines when the activity can be completed. An activity entity is associated to a non-empty set of *work item*s. A work item entity is an atomic job symbolising the place where effective work is done. Contrary to activities, there is no control flow scheduling the work item execution. A work item has a description (a textual field), a casting rule (an SQL statement) that identifies the work item performer (en identifier user), a pre-condition and a post-condition (SQL statements) which define, respectively, when it can start and complete. The parallel execution of activities or work items enables data to be shared when executing (objects, rows and files) and provides flexibility and cooperation. However, parallel execution consistency control concerning data access is the responsibility of programmers. *WCO* workflow model emphasises control flow but does not describe data flow between either activities or work items. *WCO* separates the process definition model (including process control flow definition) and the process execution context (including casting rules definitions, used application information, condition definitions and rule definitions, etc.). While the former is encapsulated in the *WCO* framework, the latter is handled by external database management systems (Hitachi or Oracle RDBMS). Thanks to this flexibility of *WCO* architecture, the modification of a process execution context is possible without process definition model alteration. Moreover, *WCO* maintains the possibility of modifying process definitions at runtime. Thus, cohabitation of several versions of the same processes can be ensured. Besides these capabilities of definition and enacting processes, *WCO* offers facilities to monitor life cycles of process instances, activities and work items. Finally, it integrates a CORBA Workflow Application Programming Interface (or WAPI) which is *WfMC*

Interface 2 compliant. This interface allows external applications to invoke some of the workflow methods and functionalities. For this purpose, the *WorkCoordinator* is based on a *Visigenic broker Architecture for C++*.

3.2 CORVETTE software design

The constraint of not modifying the *WorkCoordinator* system source code has seriously directed CORVETTE design choices. This accelerated our choice for cooperative process modelling and simplified discussions about architecture. However, the scope of cooperation model to be implemented in CORVETTE, has been enough limited. Otherwise, we have clearly distinguished between what is the content of workflow services and what is the content of data flow and concurrency services, which was an important policy matter in middleware software programming.

CORVETTE cooperative process modelling. The question is: how to model cooperative processes? In the ECOO project, we study two complementary approaches. The first approach consists in providing new workflow operators to explicitly point out where cooperation is possible [18]. These 'cooperative operators' extend the set of traditional operators [32]. The second consists in modelling a cooperative process in the same way as traditional production processes, using the same set of operators, but interpreted in another way (i.e. a cooperative way). In other terms, in a traditional interpretation, activities are seen as black boxes, while in a cooperative interpretation they are seen as white boxes (being allowed to share intermediate results).

In CORVETTE, we chose the second alternative, which has two qualities:

- First, it corresponds to (a) reality in process books, administration processes are described in the same way as design processes. Actually, the workflow process reader interpretation changes depending on his/her know-how of application domain (idea depicted in Figure 2).
- Second, introducing cooperation has no impact on process modelling and allows us to reuse *WCO* process modeller, to model CORVETTE cooperative processes, as it is. Thus, this respects the constraint of not modifying the *WCO* source code, but transfers the problem to process management (i.e. enactment, monitoring, etc.).

Another decision was, without losing generality, to have *one work item per activity*.

CORVETTE cooperation patterns modelling. Based on the decision of keeping the *WCO* model for cooperative process modelling and the constraint of not modifying the *WCO* source code, a question that arises is: which cooperation, or which cooperation patterns, is still possible to model with this constraint? In fact, and fortunately, *WCO* is mainly concerned with control flow and is very permissive regarding data flow. In fact, it imposes no constraint on data flows and cooperation between two work items, and in our context between two parallel activities, is allowed. An important restriction came, however, from the fact that a *WCO* activity can execute only when its preceding activities have terminated their executions. This means that only activities in parallel branches can cooperate, which has an impact on cooperative process modelling. Moreover, cooperation patterns *Producer/consumer* and *Redactor/reviewer*, referring by nature to succeeding activities, cannot effectively be implemented. In other terms, either these patterns are not allowed, or their implementation implies some contra-nature modelling.

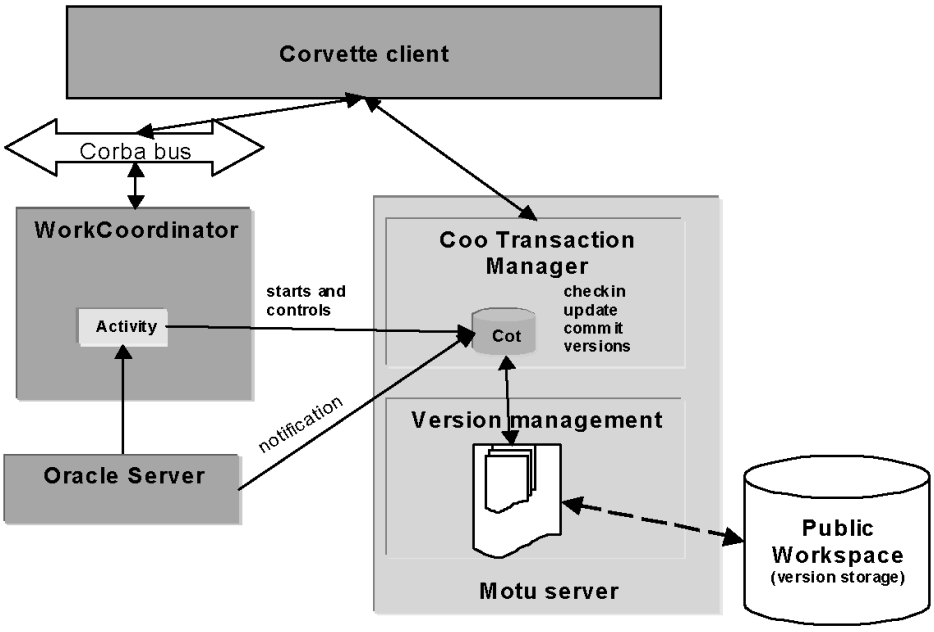
3.3 CORVETTE software architecture

CORVETTE software components. The *WorkCoordinator* workflow management system handles workflow process modelling, enactment and coordination of workflow activities delegating data management services to the programmer responsibility. The *MOTU cooperative system* supports COO-transactions entities and manages a versioned document workspace for each COO-transaction. The CORVETTE architecture, thus, integrates two main components: a workflow management component handling control flow between workflow activities and a data management component handling data versioning and data flow between COO-transactions.

In this architecture, it was not possible to modify the *WorkCoordinator*, so the decision was quickly oriented towards the definition of a mediator between *WorkCoordinator*, *Motu* and users without intervening in contributing components source code. However, this was not the single reason for choosing this architecture. We think that clearly distinguishing between a component for control flow management and another for data flow management involves considerable software architecture in middleware programming context.

Figure 3 depicts the overall CORVETTE architecture. A *CORVETTE Client* component assumes the mediator role between *one user*, *WorkCoordinator Server, and Motu Server* (embedding *COO-transactions and workspaces Managers*). In other words, there is one *CORVETTE Client* per user. This latter can simultaneously be performer of several work items, executing in different workspaces. Concurrency management between work items that are interfaced by one or several *CORVETTE Client*s leads to *Motu COO-transaction Manager*. More information on CORVETTE implementation details are given in [33].

Figure 3 CORVETTE general design



CORVETTE plugging rules. The main issue in defining *CORVETTE Client* was the ability to establish a correspondence and coherency between *WorkCoordinator* process definition entities (handled by *WorkCoordinator Server* and *Oracle Server*) and COO-transactions entities (handled by *MOTU Server*). In other terms, how does one encapsulate a *WorkCoordinator* unit of work in a COO-transaction entity?

As, in *WorkCoordinator*, real work is executed in work items, the decision was to associate each work item to a COO-transaction. As work items can run in parallel, cooperation in the sense of intermediate result sharing can occur. And encapsulating work items in COO-transaction asserts consistency of cooperation in the sense of the COO protocol. As work items are started automatically by the Work Coordinator execution engine as soon as their activation conditions are fulfilled (e.g. the preceding work item has completed), this must be detected in order to create the corresponding COO-transaction. Reciprocally, termination of a work item must be done in coordination with its corresponding COO-transaction. This means that termination condition of the work item and of the COO-transaction must be fulfilled at the same time. Another issue concerns termination of a group of COO-transactions. When several COO-transactions are grouped due to cyclic dependences between them, they must terminate simultaneously, following a kind of two phases termination protocol. That means that all corresponding work items must also terminate simultaneously. To implement this capability, one new state was introduced (*WaitingCommit*) in the COO-transaction model. Table 1 depicts this mapping.

Table 1 WorkCoordinator work item/COO-transaction states mapping

Work item state	*COO-transaction state*
Initial	Initial
Performing	Executing
	Waiting Commit
	RTC
Completed	Terminated

The role of CORVETTE client is to manage this mapping. This is mainly performed in the *create work it*em, *perform work item, open workspace* and *terminate work item* CORVETTE commands as follows:

- *CORVETTE create work item*: this command overwrites the *WorkCoodinator create work item* command to create a work item, and its associated COO-transaction structures, including the corresponding private workspace
- *CORVETTE perform work item*: this command overwrites the *WorkCoordinator perform work item* command to manage the associated COO-transaction (pushing it in *executing* state)
- *CORVETTE open workspace*: this command allows to create and populate the private workspace associated to the COO-transaction with necessary work item enactment artefacts
- *CORVETTE terminate work item*: this command overwrites the *WorkCoordinator terminate work item* command to manage the associated COO-transactions. Let us explain the algorithm of this command:

Let t_w this work item associated COO-transaction.

1 if t_w is not in a group of COO-transactions:
  1.1 if t_w is dependent on another one (has read an intermediate result of another active COO-transaction), it must wait for this other transaction to terminate
  1.2 if not, t_w enters the *waiting commit* state and asks its encapsulated work item to terminate
    1.2.1 if the work item terminates (all its termination conditions are fulfilled), the COO-transaction t_w commits
    1.2.2 if not, the COO-transaction t_w returns to the *executing* state

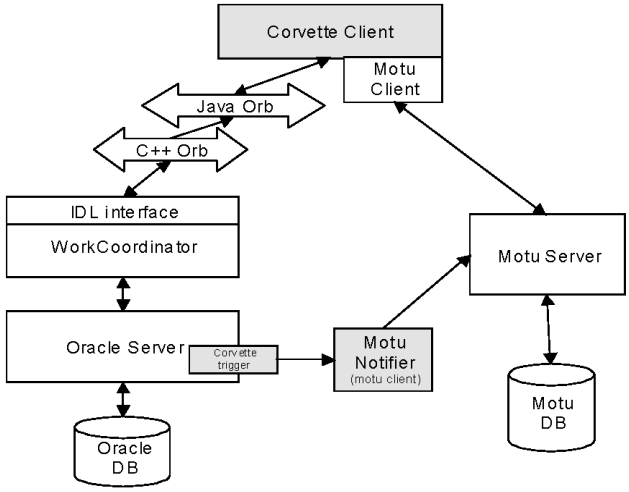
2 if t_w is member of a COO-transaction group:
  2.1 if there are still COO-transaction(s) of the group *executing*, the COO-transaction t_w enters the *ready to commit* (*RTC*) state
  2.2 if all other grouped COO-transactions are in *RTC* state, the COO-transaction t_w triggers the termination of all encapsulated work items of the group and enters the *waiting commit* state
    2.2.1 if all group work items terminate, all their associated COO-transactions are committed
    2.2.2 if not, as a work item cannot go back to the *performing* state, a human group decision session has to be launched to insure a manual recovery of the work item group.

As a direct consequence of cooperation cycles detection, the problem with a 'work item group' termination is that all the group work items, associated to grouped COO-transactions, have to terminate simultaneously. A termination conflict occurs when a sub-set of such a group have terminated and that one other work item of this group cannot. As other work items are not able to return from *terminated* to *performing* state, the system is blocked. In the context of our applications, *abort* is not acceptable and direct human intervention for manual recovery of the work item group is necessary. As in other computer supported cooperative applications, human decision is needed to help the system to solve conflicts (e.g. exception recovery, resolution of indeterminism, negotiation of needed set of values [34], etc.). *WorkCoordinator* experts affirm, however, that the case where a work item fails is extremely rare.

4 CORVETTE implementation

As depicted in Figure 4, CORVETTE is a client for, on the first hand, the *WorkCoordinator Server*, and on the other hand, the *Motu Server*. Concerning the interface between *CORVETTE client* and *Motu server*, as both are written in Java, *CORVETTE client* is simply a special Java RMI (Remote Method Invocation) *Motu Client*. While, the interface between *CORVETTE Client* and *WorkCoordinator Server* (written in C++), it is based on *WorkCoordinator IDL interface* and on interoperability between *Visigenic ORB for C++* and *Visigenic ORB for Java*.

Figure 4 CORVETTE implementation overview



Otherwise, in order to detect work items creation and termination, as necessary for transaction management, triggers and Java stored procedures have been added to the *WorkCoordinator* workflow relevant data managed by an *Oracle Server* (*CORVETTE trigger* component). Each time a new work item is created or deleted, a message is sent to the *Motu notifier* component. Note that the installation of these triggers in the *WorkCoordinator* relevant data database is the only intervention done in some *WorkCoordinator* structures.

Finally, the *Motu* notifier is a special *Motu Server* that monitors *WorkCoordinator* events and creates a transaction each time a work item is started. It manages also notification information for users awareness support.

5 Synthesis and conclusion

Globally, this experiment is a success. We demonstrate the feasibility of defining a cooperative workflow management system by 'plugging' (in the sense of integration software component without modification) together a 'traditional' workflow management system and an advanced cooperative transaction model. The main reason for this success is the absence of data flow consideration in the *WorkCoordinator*. Thus, we did not have to manage the integration of *WorkCoordinator* data flow model with our transaction model. Another success is the demonstration of the ability to model cooperative processes as traditional processes, but to interpret them in a cooperative way corresponding to cooperative behaviours.

This success is limited in the sense that not all cooperation capabilities, initially forecasted, have been implemented in such a flexible way (due to the inability to share intermediate results between succeeding activities). To overcome this limitation, it is necessary to provide activities with the capability to anticipate:

> "anticipation is the weakening of strict sequential execution of activity sequences in a process by allowing intermediate results to be used as preliminary input of succeeding activities."

For more about anticipation, see [35]. Anticipation allows the implementation of *Producer/consumer* and *Redactor/reviewer* between succeeding activities, thus providing support for the full ECOO cooperation model.

As a conclusion of this experiment, we think that, if a workflow manager component does not impose constraints on data flow, and if it provides the capabilities introduced in the previous paragraph (anticipation, events and group termination), it will be possible to completely develop a cooperative workflow management system by simply plugging together this workflow component and a cooperative transaction manager. In addition, if all the activities of the process are concurrent (i.e. execute in isolation) as the process model does not change, this workflow management system has the behaviour of a traditional (competitive) one.

The experience gained in the Corvette project deeply influences our current developments, namelythe *Bonita* [36] flexible workflow management system and the *Toxic Farm* [37] portal for virtual team hosting.

References and Notes

1 CORVETTE stands for Coordination of a Virtual Team.

2 Agostini, A. and De Michelis, G. (1996) *Modeling the Document Flow within a Cooperative Process as a Resource for Action*, University of Milano.

3 Baker, D., Georgakopoulos, D., Schuster, H. and Cichocki, A. (2002) 'Awareness provisioning in collaboration management', in *International Journal of Cooperative Information Systems.*

4 Schuster, H., Baker, D., Cichocki, A., Georgakopoulos, D. Rusinkiewicz, M. (2000) 'The collaboration management infrastructure', in *International Conference on Data Engineering* (ICDE), San Diego.

5 Grigori, D., Charoy, F. and Godart, C. (2001) 'Flexible data management and execution to support cooperative workflow: the COO approach', in *The Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS)*, IEEE Press, Beijing, China.

6 Baïna, K., Benali, K. and Godart, C. (2001) 'A process service model for dynamic enterprise process interconnection', in *9th Int. Conf. on Cooperative Information Systems (CoopIS)*, Springer Verlag, Trento, Italy, LNCS 2172.

7 Baïna, K. and Dustdar, S. (2002) 'Web-services coordination models', in *Second International Workshop on Cooperative Internet Computing (CIC)*, Kluwer Academic Publishers, Hong Kong.

8 Reuter, A.a.S., F. (1995) 'Contracts – a low-level mechanism for building general-purpose workflow management systems', *IEEE Data Engineering Bulletin*, Vol. 18, No. 1.

9 Suchmann, L.A. (1987) *Plans and Situated Action. The Problem of Human-Machine Communication*, Cambridge University Press.

10 Nutt, G.J. (1996) 'The evolution toward flexible workflow systems', in *Distributed Systems Engineering*.

11 Reichert, M. and Dadam, P. (1998) 'ADEPTflex – supporting dynamic changes of workflows without losing control', *Journal of Intelligent Information Systems*, Vol. 10.

12 Weske, M. and Schneider, B. (2002) 'An XML-centred system architecture for flexible electronic services', in *International Journal of Information Technology and Decision Making*, Vol. 1, No. 3, pp.525–540.

13 Ellis, C. and Maltzahn, C. (1997) 'Chautaqua workflow system', in *30th Hawaii Int Conf. On System Sciences, Information System Track.*

14 Han, Y., Sheth, A. and Bussler, C. (1998) 'A taxonomy of adaptive workflow management', in *Towards Adaptive Workflow Systems,* CSCW Workshop, Seattle, USA.

15 Casati, F. and Pozzi, G. (1999) 'Modeling exceptional behaviors in commercial workflow management systems', in *4th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS)*, IEEE Computer Society Press, Edinburgh, Scotland.

16 Joeris, G. (1999) 'Defining flexible workflow execution behaviors', in *Enterprise-wide and Cross-enterprise Workflow Management – Concepts, Systems, Applications*, *GI Workshop Proceedings – Informatik*, Ulmer Informatik Berichte Nr. 99-07, University of Ulm.

17 Baïna, K. *et al.* (2002) 'CORVETTE: a cooperative workflow development experiment', in *3rd IFIP Working Conference on Collaborative Business Ecosystems and Virtual Enterprises (PRO-VE'02)*, Kluwer Academic Publishers, Sesimbra, Portugal.

18 Godart, C. (1993) 'COO: a transaction model to support cooperating software developers', in *European Software Engineering Conference*, Garmisch, Germany, LNCS 717.

19 Godart, C., Olivier, P. and Skaf, H. (1999) 'COO: a workflow operator to improve cooperation modeling in virtual processes', in *Research Issues in Data Engineering*, Sydney, Australia.

20 WfMC, *Workflow Management Coalition*, http://www.wfmc.org/

21 Jablonski, S. and Bussler, C. (1996) *Workflow Management – Modeling Concepts, Architecture and Implementation*, International Thomson Computer Press.

22 Leymann, F. and Roller, D. (1999) *Production Workflow*, Prentice Hall.

23 WCO, *Hitachi-WorkCoordinator,* http://www.hitachi.co.jp/Prod/comp/soft1/wco/eng/index.html

24 Motu, http://motu.sourceforge.net

25 COO stands for COOperation transaction model

26 BSCW, *Basic Support for Cooperative Work*, http://bscw.gmd.de/

27 Teamscope, http://www.teamscope.com/

28 Sourceforge, http://www.sourceforge.net

29 Molli, P., Skaf-Molli, H. and Bouthier, C. (2001) 'State tree map: an awareness widget for nultisSynchronous groupware', in *7th International Workshop on Groupware*, CRIWG, IEEE Computer Society, Darmstadt, Germany.

30 Canals, G., Charoy, F., Godart, C., Molli, P. and Munier, M.A. (1998) 'Criterion to enforce correctness of indirectly cooperating applications', in *Information Sciences: an International Journal,* Vol. 110*,* pp.279–303.

31 Skaf, H., Charoy, F. and Godart, C. (1999) 'Maintaining shared workspaces consistency during software development', *Software Engineering and Knowledge Engineering Journal*, Vol. 9, No. 5.

32 Typically and-join, and-split, or-join, or-split, sequence as defined by W*f*MC.

33 CorvetteManual (2001) http://www.loria.fr/equipes/ecoo/corvette/CorvetteUserManual.htm.

34 Munier, M., Baïna, K. and Benali., K. (2001) 'A negotiation model for CSCW', in *5th International Conference on Cooperative Information Systems (CoopIS)*, Springer Verlag, Eilat, Israel, LNCS 1901.

35 Grigori, D., Charoy, F. and Godart, C. (2001) 'Anticipation to enhance flexibility of workflow execution', in *DEXA Conference*, Munich, LNCS 2113.

36 Bonita (2003) http://bonita.debian-sf.objectweb.org/.

37 Toxic Farm (2003) http://woinville.loria.fr.