# A Pattern-based Process Management System to Flexibly Execute Collaborative Tasks

Mamadou Lakhassane Cisse[1,3], Hanh Nhi Tran[2], Samba Diaw[3], Bernard Coulette[1] and Alassane Bah[3]
[1]*IRIT Laboratory, Jean Jaures University, Toulouse, France*
[2]*IRIT Laboratory, Paul Sabatier University, Toulouse, France*
[3]*UMMISCO, Cheikh Anta Diop University, Dakar, Senegal*

Keywords: Process Management, Process Execution, Multi-instance Task, Collaboration Process Pattern, Late-binding.

Abstract: Managing collaborations during the execution of a process is complex and challenging, especially for managing the collaboration inside a task having multiple instances performed by various actors. Existing process management approaches propose either a rigid control for conducting such collaborative tasks or no control at all. Aiming at a more flexible way to execute and control multi-instance tasks, we investigate a solution based on late-binding mechanism to allow actors choosing dynamically suitable strategies to perform their collaboration. First, we propose a process modeling language which focuses on describing multi-instance tasks and their dynamic instances at execution time. Second, this language is then used to represent patterns capturing reusable collaboration strategies. A prototype of a pattern-based Process Management System has been developed to demonstrate the possible flexible execution of collaborative tasks.

## 1 INTRODUCTION

Collaborative processes have to deal with dynamic and complicated relationships between participants of a given project. Thus, they should benefit from process management methods and tools which, based on the information flows described in the process model, allow coordinating the activities performed by process actors at process execution time. Conventionally, a task is the smallest unit of work in a process subject to management accountability. Existing process management systems focus on coordinating process's tasks but pay less attention to managing the collaboration of actors inside a given task to achieve a common goal. In general, there are two usual approaches to deal with the collaboration inside a task: (1) representing the collaborative task only in the process model but ignoring interactions among actors during its enactment; (2) refining the inner steps inside the task concerning actors and describing these sub-tasks in the process model so that they are manageable during process execution. In the first case, the collaboration at runtime is not monitored and controlled, in the second case, the collaboration is controlled but is rigid and cannot adapt to evolving contexts, for example when the number of task's actors changes.

Many works have been done on designing collaboration strategies (Briggs et al., 2006; Kolfschoten and de Vreede, 2007), on modeling collaborations (Lonchamp, 1998; Hawryszkiewycz, 2005; Kedji et al., 2012; Antunes et al., 2013; Gallardo et al., 2013). However, there have been few studies about assisting and controlling the execution of collaborative processes, especially to allow the collaboration to evolve according to the process application context (Ariouat et al., 2016).

Motivated by this lack when working on modeling and execution of collaborative processes (Cisse et al., 2018), we have been investigating a solution for executing collaborative tasks in a flexible way. We are interested in a special form of collaborative tasks, so-called Multi-Instances Tasks (MIT). At execution time, a single task is instantiated once and the task instance is performed by an actor playing the required role. An MIT needs several instances to accomplish the task's goal. The relationships among the instances of an MIT depend on the collaboration strategy used to realize the task's goal. In order to enable a fine control of an MIT's enactment, these inter-instances relations must be defined.

As an extension of the work in (Cisse et al., 2018), this paper presents our proposition to allow executing flexibly multi-instance tasks. The main steps are: (1) defining a set of patterns to capture different collab-



Cisse, M., Tran, H., Diaw, S., Coulette, B. and Bah, A.
A Pattern-based Process Management System to Flexibly Execute Collaborative Tasks.
DOI: 10.5220/0007680702730280
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 273-280
ISBN: 978-989-758-375-9

oration strategies that define typical relations at execution time among the instances of a multi-instance task in specific situations; (2) using the patterns in (1) to specify flexibly the inter-instances relations of an MIT at execution time in order to allow actors adapting their collaboration strategies according to the project's context.

The main contribution of this work is the proposal of the late-binding mechanism to dynamically choose the execution strategy of a multi-instance task. In order to do so, we have defined a set of collaboration patterns from which one is chosen at execution to conduct a particular strategy in a given project's context.

A collaboration pattern captures a recurrent collaboration strategy that can be used at execution time to perform a collaborative task in a specific context of the application project. A collaboration pattern describes the typical relationships among the instances of a collaborative task from two main perspectives: the control-flow and the data-flow. The control-flow perspective provides the execution order of task instances, represented by the work-sequence relations among the instances. The data-flow perspective specifies, via the task parameter relations, the data manipulated, exchanged or shared by the task's instances. In contrast to the modeling patterns proposed in (Lonchamp, 1998), (der Aalst et al., 2003) and (Vo et al., 2015) that are applied at modeling time for describing collaboration scenarios, our collaboration patterns (see Section 2.2) are applied dynamically at execution time to generate the detailed model of running collaborative tasks.

A loosely specified collaborative task at modeling time needs to be completed at execution time to enable a controlled flexible execution. To do so, we use the late-binding mechanism to select dynamically a suitable collaboration pattern and use it as a template to generate the inter-instances relations for the collaborative task. To enable adapting the execution of a collaborative task to the evolution of a project context or to change collaboration strategy during the execution, a collaborative task can be bound to different collaboration patterns. Generally, the main factors that can impact the choice of a collaboration are constraints on the availability of resources (humans, time, tools etc.) or constraints on the order of manipulating or producing the inputs and outputs. Selecting (semi-) automatically a suitable pattern is out of scope of this paper.

Furthermore we have also defined an operational semantics that enables the application of collaboration patterns to make the execution of collaborative tasks flexible. Due to space constraint, this semantics is not presented here.

The paper is structured as follow. Section 2 presents first our executable process modeling language used to describe collaborative tasks and the relations among their instances at execution time, and second collaboration patterns for MITs execution. A prototype implementing our work is described in Section 3. We discuss in Section 4 some related works and summarize in Section 5 our contributions as well as our perspectives.

# 2 MODELING COLLABORATIVE TASKS

To enable a fine-grained control of collaboration during process execution, both the structural and behavioral aspects of collaborative tasks must be known. Aiming at a flexible execution, we define these aspects of a collaborative task in two times: at modeling time, only the task's structural elements (e.g. performing role, used artifacts) are described, then at execution time, when the task is instantiated into several instances, the relations among the task's instances (e.g. work-sequences, exchanged-data) will be specified according to the used collaboration strategy and in this way reflect the task's behavior. The challenge for such an approach is how to generate at execution time the behavioral model of a collaborative task without requiring process actors to go back to the modeling phase.

Dealing with this issue, first in Section 2.1 we propose the *Executable Collaborative Process Modeling Language (ECPML)* which is composed of two packages allowing to represent the process elements at modeling time (ECPML_Core ) and their dynamic instances at execution time (ECPML_Execution). Then in Section 2.2 we propose a catalog of collaboration patterns that can be bound dynamically to a collaborative task instance at execution time to define its behavior.

## 2.1 Executable Collaborative Process Modeling Language

Figure 1 shows an extract of the meta-model of our *Executable Collaborative Process Modeling Language (ECPML)*. This language is inspired from SPEM (OMG, 2008) for the elements of ECPML_Core. The concepts representing ECPML_Execution were defined by ourselves because SPEM does not provide elements concerning process execution.

The Figure 1a defines the process elements and

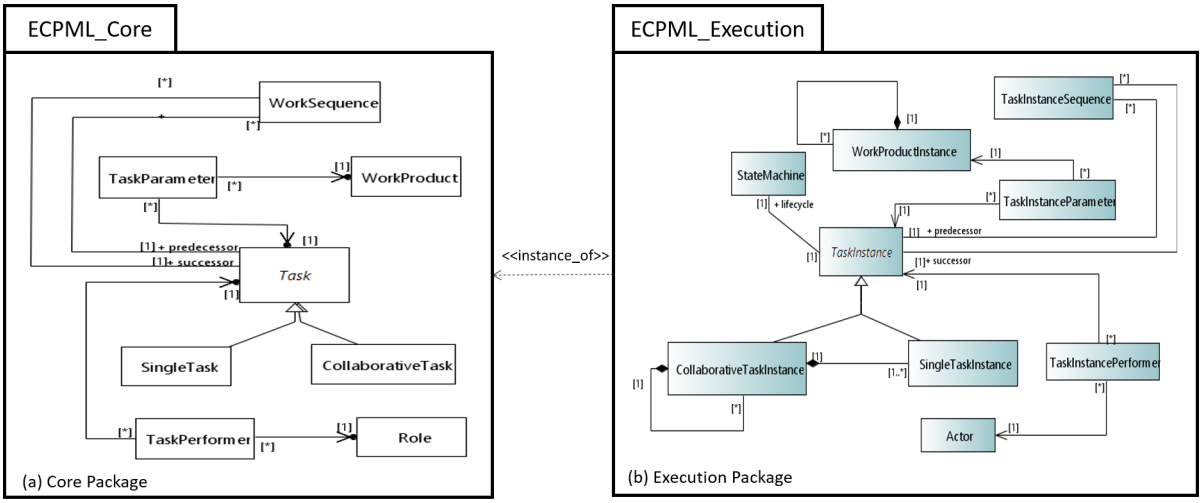


Figure 1: Extract of the meta-model defining the ECPML. "instance_of" dependency abstracts the instantiation links between concepts of ECPML_Core and ECPML_Execution.

the relations among them. Our main focus in this paper is the *Task* concept representing a manageable unit of work. A *Role* is an abstract entity representing some qualifications. The relation *TaskPerformer* allows to specify a specific role required to perform a given task. *WorkProduct* represents concrete, tangible entities used or produced during the execution of a task via the relation *TaskParameter*. The relation *WorkSequence* between two tasks represents the constraint on the orders to execute the tasks. Our language distinguishes two types of task: a *SingleTask* which has only one instance at execution time and a *CollaborativeTask* which can have several instances at execution time.

The Figure 1b defines the concepts used to represent the execution of a process, i.e. the dynamic instances created at execution time from the elements defined at modeling time in a process model. The relation *instance_of* represents the mapping between the two packages of the meta-model. As an example we can say: a *Task* at modeling time is instantiated at execution time into one or many *TaskInstances*; a *WorkProduct* at modeling time is instantiated at execution time into one or many *WorkProductInstances*. A *Role* is played by one or many *Actors* at execution time. The relations between the instances are the same as those defined between their modeling concepts. Concretely, a *TaskInstance* can have *TaskInstanceParameter* relations with the *WorkProductInstances* that it uses or produces, a *TaskInstance* has a *TaskInstancePerformer* relation with the *Actor* who enacts it and a *TaskInstance* can have *TaskInstanceSequence* relations with other task instances to specify their execution orders.

Focusing on controlling the execution of an MIT performed by several actors, we distinguish *SingleTaskInstance (STI)*, which is an instance of *SingleTask*, and *CollaborativeTaskInstance (CTI)*, which is an instance of *CollaborativeTask*. An *STI* is the main executable element representing a unit of work assignable to a single actor. A *CTI* is composed of several *STIs* performed by separate actors.

We present in Figure 2 the example *Writing* process modeled in ECPML. It contains two single tasks *WriteDocument* and *ReviseDocument*, each one being performed by one actor playing the role *Author*, and a collaborative task *ReviewDocument* which is performed by several actors playing the role *Reviewer*. *Manuscript* and *Assessment* represent the in and out *WorkProduct* of the tasks.

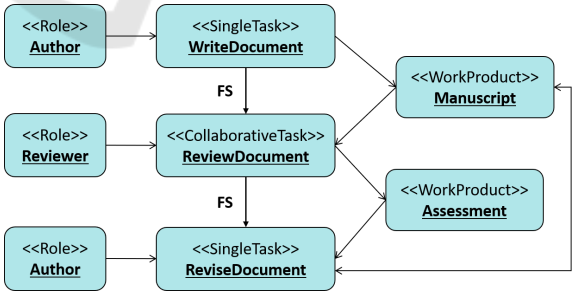


Figure 2: Model of the Writing process in ECPML at modeling time.

Controlling the execution of a process is essentially coordinating the execution of different *STIs*. To do so, the *TaskInstanceSequence* relations among the *TaskInstances* must be known. However, only those specifying the relations among the instance of different tasks are defined in the process model and thus known at modeling time. For example, from the pro-

cess model in Fig. 2, we can know only the work-sequences *FS* (i.e. *Finish2Start*) between the *WriteDocument* and the *ReviewDocument*, between the *ReviewDocument* and the *ReviseDocument*. However, the relations among multiple instances of a collaborative task, i.e. among the *STIs* inside a *CTI* as among the *STIs* of the *ReviewDocument CTI*, are not described in the process model and need to be established at execution time.

The inter-instances relations among the *STIs* of a *CTI* are dependent on the collaboration strategy used to carry out the task. In our approach, we use collaboration patterns to capture such strategies and to reuse these patterns as templates to define the relations between the *STIs* inside a *CTI*. The following section presents how ECPML can be used to model collaboration patterns and shows two representative patterns.

## 2.2 Patterns for Multi-instances Task Execution

We have identified several patterns based on the way the manipulated artifacts are shared. In this paper, we consider only how the output artifacts that are changed by the collaborative task are shared among its instances. The input artifacts are implicitly considered as *"read-only"* items shared by the instances inside the collaborative task and are not shown in the patterns.

Using our own template, a pattern is presented with its name, description, problematic, context of use, and a solution described in ECPML. Presenting the exhaustive list of these patterns is out of scope of this paper. In the following, we present 2 representative patterns corresponding to the two main types of execution: in parallel and in sequence.

### 2.2.1 Pattern Parallel Instances with Composite Out Parameter

**Name:** PAR-INSTANCES-COP
**Problematic:** need to realize a quick production of a composite artifact made of independent parts
**Context:** This pattern serves to manipulate a composite artifact regardless of the order of production for the different parts. It means that those parts must be independent. It can be used when all the resources needed for the enactment of the collaborative task are available.
**Description:** Given a collaborative task $T$ having one output parameter $P$ composed of $n$ independent components $P_i$, this pattern is used to execute a set of $n$ single task instances $t_i$ inside the collaborative task instance of $T$ simultaneously. Each task instance $t_i$ will manipulate separately an instance $p_i$ of the component $P_i$. The different task instances can be executed at any time as there is no sequencing between them. They should be executed in parallel to minimize the execution time.
**Solution:** Figure 3 shows the models of *PAR-INSTANCES-COP* pattern respectively at modeling time, and at execution time with 2 task instances of the collaborative task $T$.

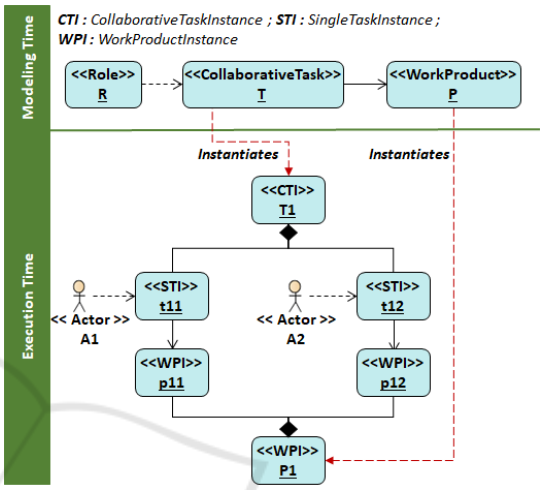


Figure 3: Pattern PAR-INSTANCES-COP for a collaborative task T.

### 2.2.2 Pattern Sequential Instances with Composite out Parameter

**Name:** SEQ-INSTANCES-COP
**Problematic:** need for a progressive production of a composite artifact made of dependent parts.
**Context:** This pattern serves to manipulate different components of a composite artifact in a sequential order determined by the dependencies among them. This can happen when we need to divide the work on the production of the composite artifact among several actors according to their availability.
**Description:** Given a collaborative task $T$ having one output $P$ which is composed of $n$ dependent components $P_i$, $i \in [1,n]$, this pattern is used to execute a series of $n$ consecutive single task instances $t_i$, $i \in [1,n]$ inside the collaborative task instance of $T$. Each task instance $t_i$ manipulating an instance $p_i$ of the component $P_i$ and is performed by a different actor playing the same role. The execution order $FS$ among the task instances is imposed by the dependencies defined among the components of $P$: the creation of $P_i+1$ needs the completion of $P_i$ thus $t_i+1$ (which works on $P_i+1$) has to follow $t_i$ (which produces $P_i$). The value of $n$ can be given when the collaborative task is deployed.

**Solution:** Figure 4 shows the models of *SEQ-INSTANCES-COP* pattern respectively at modeling time, and at execution time with 2 task instances of the collaborative task $T$.

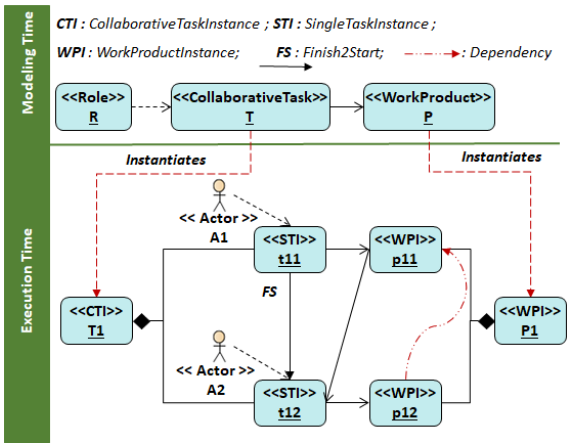


Figure 4: Pattern SEQ-INSTANCES-COP for a collaborative task T.

As a dynamic entity, a *TaskInstance* has a lifecycle composed of different states through which it goes when executed. To allow deploying and executing a multi-instance task, we need to define the task instance's lifecycle, its operational semantics. As mentioned above, the semantics of the behavior of the process engine is out of scope of this paper.

Figure 5 describes the application of the pattern SEQ-INSTANCES-COP presented in Figure 4 to describe the behavior of the collaborative task *Review*. Notice that the resulting Review document is composed of Alice Assessment and Bob Assessment.

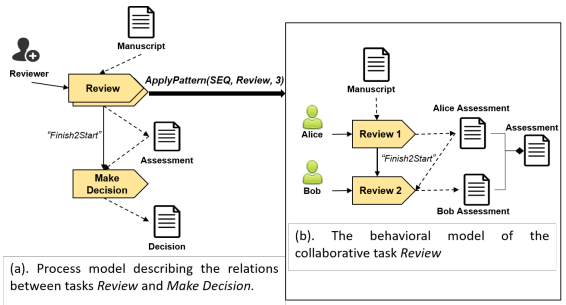


Figure 5: Application of SEQ-INSTANCES-COP to define the relations among the task Review's instances.

## 3 IMPLEMENTATION OF A PROTOTYPE

As a proof of concept, we have developed the prototype *CPE (Collaborative Process Engine)*, a process management system supporting flexible execution of multi-instance tasks so that users can choose, at enactment time, appropriate collaboration strategies corresponding to their organizational model. Figure 6 below describes the general architecture of CPE.

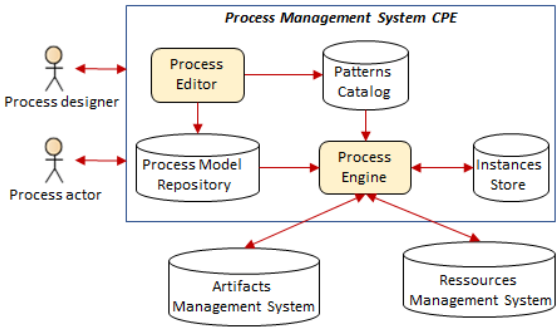


Figure 6: General architecture of CPE.

CPE has two main components:

- **Process Editor** allows process designers using ECPML to model processes and collaboration patterns then store them into *Process Model Repository* and *Patterns Catalog*.
- **Process Engine** supports process actors performing their process, i.e. instantiating the tasks defined in their process model and managing the execution of these tasks. Process actors choose a process model in *Process Models Repository* to execute and the process engine will generate the dynamic instances of the process's elements and store them in the *Instances Store*. At execution time, the process engine updates the process's instances, by using the operational semantics, to evolve the process. To deploy a collaborative task, the number of instances for each task can be given by the project manager or imposed by the organizational model of the project. Then according to the project characteristics, the project manager chooses the most suitable collaboration pattern to the project's context, among those of the *Patterns Catalog*, for executing each collaborative task.

The physical artifacts and human resources manipulated during the process execution are managed by external *Databases*: *Artifacts Management System* for artifacts and *Resources Management System* for actors. These databases are connected to CPE which manages just the references of artifacts and actors inside its *InstancesStore*.

Figure 7 shows a screenshot of the execution of our motivating example in the prototype. In this step, the manager chooses the most suitable pattern to deploy the collaborative task *Review*. Given that all the actors are available at the same time and the different

document's parts to be reviewed are independent, he has chosen the pattern *Parallel Instances Composite Out Parameter*.

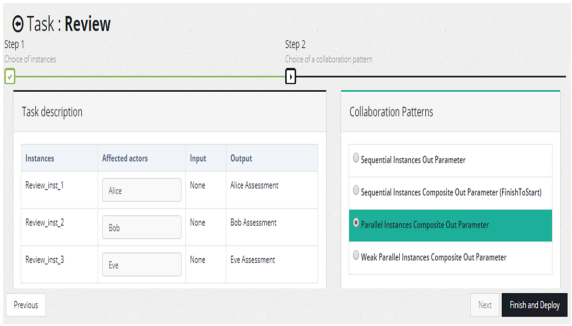


Figure 7: Screenshot of the CPE prototype.

Thanks to CPE, the project manager can monitor the execution of collaborative tasks and adapt the collaboration strategy for conducting collaborative tasks at any moment according to the alteration of project's constraints and needs. CPE provides process actors with not only the necessary functionalities to perform their task (by verifying the condition to create, start, end or assign resources to a task instance) but also a global and real-time view on the progress of development tasks (by showing the information about the collaborative task that he participates in: what is the current state, who are other actors performing the task, what are exchanged data, etc.).

Although CPE is helpful for all kinds of processes which have multi-instance tasks, it can benefit particularly system and software processes which are often performed by several teams to produce different parts of the final product. Moreover, generally system and software processes' projects have changing contexts because of the evolution of product's specification as well as the evolution of production's constraints. The above characteristics make system and software processes require more assistance and control during their execution - what is offered by our CPE.

## 4 RELATED WORKS

There are some works (Hawryszkiewycz, 2005; Kedji et al., 2012; Antunes et al., 2013; Gallardo et al., 2013) addressing the modeling of a collaborative process. Mostly they proposed constructs to model activities that need to be coordinated during the process execution but did not deal with the management of collaborative tasks during execution.

Process patterns have been used to describe a set of activities realized to solve common problem in a software development lifecycle or in a business process. The works in (Lonchamp, 1998; A. de Moor, 2006; Verginadis et al., 2010) proposed patterns capturing best practices about organizing collaborative activities. The mentioned works also focused only on representing collaborative strategies at modeling-time contrarily to our approach which studies the application of collaborative strategies.

The work of (der Aalst et al., 2003) proposed a series of workflow patterns representing typical solution for modeling and implementing processes. Their work is used principally for evaluating the expressiveness of process modeling languages and the supporting capacity of process management systems. In contrast to us, they did not provide a formalized solution to integrate the proposed patterns into a process system.

Few works on multi-instance tasks have also been investigated in the literature. In (Atwood, 2006), the author proposed a way to represent loop of tasks or independent tasks. However, it focuses only on control-flow between instances but ignores data-flow. (Sun et al., 2006) also introduced patterns for multiple instances activities. They dealt with the flexible execution by allowing a workflow management system to perform an activity several times, not like us who proposes defining the behavior of the system at execution time. Although (der Aalst et al., 2003) presented some workflow patterns concerning the implementation of multi-instance tasks, it does not address the question of using dynamically patterns to support a flexible execution of collaborative as done in our work.

The flexibility of process execution is an important challenge of the process community. One approach to handle this is allowing deviations inside process environments i.e., detecting, tolerating and managing deviations, as done in (da Silva et al., 2011; Smatti and Nacer, 2014). Another approach is allowing late-modeling or late-binding, i.e. the ability to deal with unpredictable situations by allowing the process model to be partially unknown at design-time and refined at run-time. As for (Dustdar, 2004), they proposed a process-aware CSCW system supporting process schemas that are created on-the-fly. In (Charoy et al., 2006), authors introduced a Workflow management system allowing users to modify the instance of a process, such as adding an activity. It has been taken into account in the development of the Bonita workflow management system. Compared to the cited works, we also adopt the late-binding approach, but propose to use dynamically patterns to parameterize the behavior of the process engine and thus make it flexible.

In (Tran et al., 2011), the authors defined patterns for modeling process and a mechanism for applying patterns to refactoring process models. This work does not tackle collaborative tasks. (Vo et al., 2015) proposed also an approach to define and apply collaboration patterns for software development modeling. Compared to these pattern-based approaches, our work allows dynamical application of patterns for adapting the behavior of a running process at execution time.

# 5 CONCLUSION

Our current research focuses on the flexible management of collaborative processes. Our work targets the modeling and execution of collaborative tasks. The work presented in this paper considers in particular multi-instance tasks which are instantiated several times at execution and performed by different actors but all collaborating to produce a common result. The objective of this work was providing a solution to model partially multi-instance tasks and then using the late-biding mechanism to complete the tasks behavior flexibly at execution time.

The main contribution of our work is the language ECPML used to model collaborative process, both structural and behavioral aspects, at modeling and execution time. We have used ECPML to model a set of collaboration patterns describing the typical behavior models of multi-instance tasks. These patterns can be bound to the structural model of a collaborative task to complete the task information and thus allow managing collaborative tasks. The execution of the collaboration is assisted by our prototype process management system CPE.

To improve the validation of our approach, we need to apply it to other case studies and especially to real projects. Indeed, it is always better to work with real project data, but our objective is mostly to test the set of collaboration strategies at execution time. Adding new collaboration patterns is also desirable but the limited set of collaboration patterns implemented, so far, does not question the validity of our approach. The proposition of more patterns, which is one of our perspective, will not put at risk the scalability of our approach since the search function complexity of a suitable pattern is linear.

We aim also supporting more complex collaborative task behaviors. Currently, we only deal with patterns describing one kind of work-sequence relations among the single tasks instances of a collaborative task (for example *Finish2Start*). However, sometimes in practice there are several kinds of inter-instances relations inside a task. To support more complex collaborations, we intend to investigate the proposition of new patterns covering those situations. We explore also the capacity of combining dynamically collaboration patterns at execution time.

One of our perspective also is to investigate the possibility of allowing a single task instance inside a collaborative task instance to become itself a collaborative task instance during enactment. Indeed, it is needed sometimes to allow a task to be refined into several instances due to constraints (such as emergency for a faster execution).

# REFERENCES


A. de Moor, A. (2006). Community memory activation with collaboration patterns. In *3rd Prato Community Informatics Research Network Conference (CIRN 2006)*, pages 1–18, Prato, Italy.

Antunes, P., Herskovic, V., Ochoa, S. F., and Pino, J. A. (2013). Modeling highly collaborative processes. In *2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2013*, number July 2014, pages 184–189.

Ariouat, H., Andonoff, E., and Hanachi, C. (2016). Do process-based systems support emergent, collaborative and flexible processes? comparative analysis of current systems. *Procedia Computer Science*, 96:511–520.

Atwood, D. (2006). Bpm process patterns: Repeatable design for bpm process models. *BPTrends*, pages 1–20.

Briggs, R., Kolfschoten, G., and de Vreede, G. J. (2006). Defining key concepts for collaboration engineering. *Association for Information Systems, 12th Americas Conference On Information Systems, AMCIS 2006*, 1:117–124.

Charoy, F., Guabtni, A., and Faura, M. (2006). A dynamic workflow management system for coordination of cooperative activities. *Proceedings of the Business Process Management Workshop*, pages 205–216.

Cisse, M., Tran, H., Diaw, S., Coulette, B., and Bah, A. (2018). Collaborative processes management: from modeling to enacting. In *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 461–466. IEEE.

da Silva, M., Blanc, X., and Bendraou, R. (2011). Deviation management during process execution. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 528–531. IEEE Computer Society.

der Aalst, W. M. V., Hofstede, A. H. T., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns.

Dustdar, S. (2004). Caramba—a process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. *Distributed and parallel databases*, 15(1):45–66.

Gallardo, J., Bravo, C., Redondo, M. A., and Lara, J. D. (2013). Modeling collaboration protocols for collaborative modeling tools: Experiences and applications. *Journal of Visual Languages and Computing*, 24(1):10–23.

Hawryszkiewycz, I. T. (2005). A metamodel for modeling collaborative systems. *Journal of Computer Information Systems*, 45(3):63–72.

Kedji, K. A., Lbath, R., Coulette, B., Nassar, M., Baresse, L., and Racaru, F. (2012). Supporting collaborative development using process models: An integration-focused approach. In *2012 International Conference on Software and System Process (ICSSP)*, pages 120–129, Zurich, Switzerland. IEEE.

Kolfschoten, G. and de Vreede, G. (2007). The collaboration engineering approach for designing collaboration processes. In Springer, editor, *International Conference on Collaboration and Technology*, volume 4715, pages 95–110, Berlin.

Lonchamp, J. (1998). Process model patterns for collaborative work. In *15th IFIP World Computer Congress, Telecooperation Conference, Telecoop'98*, Vienna, Austria.

OMG (2008). Software & systems process engineering meta-model specification v2.0. (April):236.

Smatti, M. and Nacer, M. (2014). Dealing with deviations on software process enactment: Comparison framework. In *ICAASE*, pages 108–115.

Sun, R., Liu, G., and Shi, M. (2006). The specification of workflow activity multiple instances. In *10th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2006*, pages 1179–1183, Nanjing, China. IEEE.

Tran, H., Coulette, B., Tran, D., and Vu, M. (2011). Automatic reuse of process patterns in process modeling. In *2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*, pages 1431–1438.

Verginadis, Y., Papageorgiou, N., Apostolou, D., and Mentzas, G. (2010). A review of patterns in collaborative work. In ACM, editor, *GROUP '10 Proceedings of the 16th ACM international conference on Supporting group work*, pages 283–292, Sanibel Island, Florida, USA. ACM.

Vo, T. T., Coulette, B., Tran, H. N., and Lbath, R. (2015). An approach to define and apply collaboration process patterns for software development. In *Model-Driven Engineering and Software Development - Third International Conference, MODELSWARD 2015, Angers, France, February 9-11, 2015, Revised Selected Papers*, pages 248–262.