Multiple Instantiation in a Dynamic Workflow Environment

Adnene Guabtni and François Charoy

LORIA - INRIA - CNRS (UMR 7503) BP 239
F-54506 Vandouvre-lès-Nancy Cedex, France
{guabtni,charoy}@loria.fr

Abstract. Business processes often requires to execute a task multiple time in series or in parallel. In some workflow management systems this possibility is already supported and called "multiple instantiation". Usually the term "iteration" is used to define multiple executions in series. Nevertheless, the existing solutions impose many constraints for workflow designers and decrease flexibility. Almost all of them use new operators to represent multiple instances that are integrated in the workflow as any other workflow basic operators. This way of representation encumbers and complicates the workflow so that it's unreadable for the end user. In this article, we propose a new way of defining multiple instantiations in a workflow without using exotic operators, nor complicating the workflow itself. Our approach is based on defining sets of tasks in a dynamic workflow process. Each set contains activities that must be executed multiple times. Each set is governed by constraints making it possible to supervise the multiple executions. These sets can be nested or even overlap. We use two types of sets in this work: "parallel instance's set" for those activities that are executed multiple times in parallel. And the second type is "iterative instance's set" for those that are executed multiple times in sequence. The number of instantiations to do and the condition to iterate could be evaluated at run-time. In this paper, we also show on a real process executed in an experience how this model could have been used to ease its definition.

1 Introduction

A lot of work has been done recently on the definition of business process models for different purposes. Most of these models provide the ability to define processes using XML schemas and are based more or less on the same concepts of nodes (activities), edges (dependencies between activities) and synchronization condition. However, as it has been precisely described in [9], these models do not support well multiple instantiation of group of activities or even of single activities. Multiple instantiation means the ability to repeat several time the execution of a group of activities, either in sequence, or in parallel. Different patterns for multiple instantiation have been identified in [9], [7] and [1] and these patterns have been explicitly defined as petri net models in [10]. Workflow management systems must provide practical possibilities to realize them. Current workflow

A. Persson and J. Stirna (Eds.): CAiSE 2004, LNCS 3084, pp. 175–188, 2004.

management systems adopt very different strategies to model these patterns. It can be by introducing new operators, using sub processes or defining complex model structures. However, most of them are far from providing a simple solution to apply multiple instantiation patterns.

Executing multiple times a set of activities, in parallel or in sequence is very common. Review processes, development processes, quality assurance processes are typical cases where the number of iterations or the number of execution depends on the number of people involved, the number of object produced or some other criteria defined on the state of the objects. A workflow management system must be able to model such repetition and even to allow evolution of the parts of the process that are executed multiple times.

In this paper we present an evolution that we can apply to any model because of its generality. We choose the most basic model that gives us a simple workflow with no added constructors than the edges of control.

The first part of the paper will describe the motivation for this work and recall the multiple instantiation patterns that can be found in the process modelling literature. The second part will describe how these patterns can be implemented in current workflow management systems. The third part will describe our proposal as an extension of a simple process model with a "set oriented" multiple instantiation definition. We will then present briefly how it has been implemented in the Bonita WFMS in order to support cooperative process management.

2 Motivation and Related Work

Multiple execution of group of activities is a pattern that can be found in a lot of cooperative processes. These executions are common in such cases:

- a piece of work is split up and distributed between a group of people (writing chapters of a book, coding modules)
- a piece of work has to be done by all the people with some role (review chapters of a book, testing)
- a piece of work has to be repeated until it reaches a certain level of quality (quality assurance, testing, review/editing process)

Different multiple instantiation patterns have been precisely analysed in [9],[7] and [1]. Here, we will consider the one described in [9] because it is a generalized description. These patterns are numbered from 10 to 15 in this paper.

- 10 is arbitrary cycles or loops. Modelling loops directly in a workflow is still considered as an issue even if some systems provide some way to do it.
- 11 is implicit termination. A sub process is terminated if nothing else is to be done. Most workflow engines require a final node to specify termination of a sub process.
- 12 is multiple instantiation without synchronization. Several instances of the same activity are executed. They are not synchronized. This is supported by most workflow systems

– 13 is multiple instantiation with a priori design time knowledge. An activity has to be executed multiple times and the number of instances is known at design time.
– 14 is multiple instances with a priori runtime knowledge. An activity has to be executed multiple times and the number of instances is known at run time before they are started.
– 15 is multiple instantiation without a priori runtime knowledge. The number of instances to create depends on the execution of the instances themselves. New instances have to be created while other are still executing.

3 Multiple Instantiation and Current Workflow Models

Many workflow management systems supports some of the patterns above. The possibilities of each one of some known workflow management systems are presented in [9]. In these workflow management systems, many solutions are used. The most prevalent of them is the use of the "merge" operator. Figure 1 illustrates the use of this operator to express multiple instantiation. The number of instances can be computed before the instantiation or computed while the instantiation is working as the pattern 15. The example concerns the multiple instantiation of just one activity. When we want to use this kind of solutions to express multiple instantiation of a set of activities, we have a constraint. This constraint inflicts that the group of activities must be sequentially connected. This is a hard constraint that decreases possibilities.

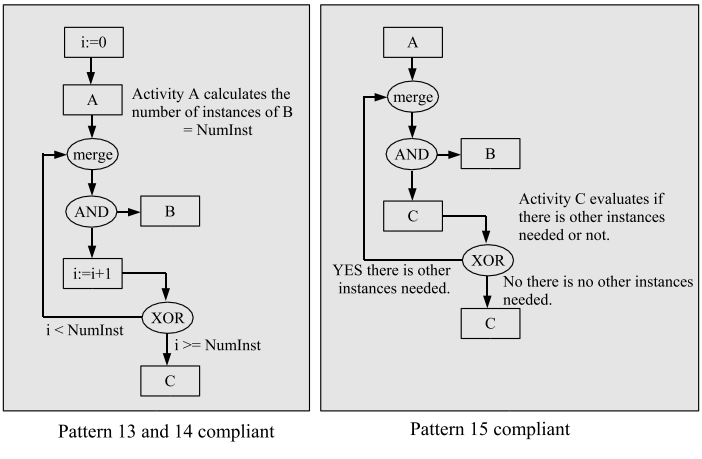

Fig. 1. Multiple instantiation using the "Merge" operator

Another kind of solution is the use of the "Bundle" operator. This solution is implemented in the FlowMark 2.3 language. The "Bundle" operator receives the number of instances of the activity it is related to and takes care of its instantiation. After that, the "Bundle" operator waits the end of all these instances

to hands off the activities that follow. Figure 2 illustrate the typical use of the "Bundle" operator in a workflow process. This solution has a disadvantage: the operator involves only one activity so that we cannot express the multiple instantiation of a group of activities.

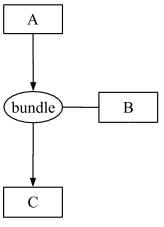

Fig. 2. Multiple instantiation using the "Bundle" operator

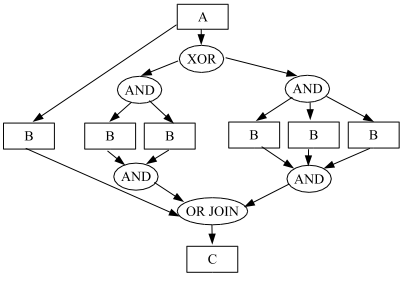

Fig. 3. Multiple instantiation using the XOR SPLIT, AND JOIN and OR JOIN operators

The workflow management systems that don't have direct support of multiple instantiation are required to use a combination of XOR SPLIT, AND JOIN and OR JOIN to do it. Figure 3 illustrates this case. We can see the complexity of the workflow process after expressing multiple instantiation. This solution supposes that the number of instances is limited to a maximum number. There is also no support for pattern 15 with this solution.

Most of all the workflow management systems are based on new operators introduction to do multiple instantiation. It can certainly offer some solutions but increase workflow structure and design complexity. The majority of these attempts cannot offer a support for all the multiple instantiation patterns.

4 Multiple Instantiation and Set Based Workflow Model

4.1 The Basic Workflow Model

Here we present an extension to workflow models. The extension firms up by adding a new kind of construction: sets. Sets are used to contain activities that have to be executed several times, either in sequence or in parallel. Set that executes in parallel is called "parallel instance set". Set that executes several times in sequence is called "iterative instance set". Both kinds of sets will have different kind of constraints for their definition and different behaviours for their execution. Sets can be nested or overlapped with certain conditions. This is what we present in the following sections.

This approach can be applied to any workflow model provided that the model allows dynamic creation of activities at runtime. Many works have been made on dynamic changes on workflow in [5], [4], [6], [2] and [5]. The Bonita workflow model offers these possibilities so that we chosen it to implement our set based model. This model is based on the work published in [2] concerning flexibility

in workflow systems and implement the anticipation of execution of activities described in [3]. Bonita model is based on a workflow model without separated process schema and instances of the schema. Our model consists of just instances. This simplifies the use of dynamic changes as the schema and the instance are the same thing.

4.2 Multiple Instantiation Sets

A multiple instantiation set of activities has a state that can be INITIAL, ACTIVE or FINISHED.

- INITIAL: All the activities of the set are in initial state (not yet executed).
- ACTIVE: At least one of the activities of the set is started.
- FINISHED: All the activities of the set have finished their execution.

There is two types of multiple instantiation sets: parallel and iterative. Each one of these two types has also some other properties and some constraints to apply on their activities.

Parallel Instance Set. Parallel instantiation deals with patterns 12 to 15 in [8]. It is a solution to execute many times the same set of activities in parallel.

The goal of parallel instance sets is to provide an answer to the problem of activities that have to be executed several times. This execution can be made in parallel. The number of times they have to be executed is not necessarily known at the beginning of the process. This is described in [8] as patterns 14 and 15.

Parallel instance set has a special property defined as:

- A function to compute the number of times it has to be executed. This function will return the number of people belonging to a role or the value of some data produced by a preceding activity.

There is no constraints on the parallel instance set of activities. Any activity that has not been yet started can be selected to participate in a parallel instance set. This allows a very simple definition of this kind of sets.

Iterative Instance Set. The goal of iterative instance sets is to allow the repetition of a their activities until some condition is evaluated to true. This is the way to model iterations in Bonita, as cycles are not allowed.

Iterative instance set matches Pattern 10 and 11 in [8]. Iteration means that we want to specify that the given set of activities must be repeated a number of times until a given condition is true. Cycles are an issue in WFMS because of race condition. In our proposal, we overcome this issue by re-instantiating activities as long as it is needed. Thus iteration is not the re-execution of a set of activities but the successive execution of copies of these activities. This has an impact on the way data are managed and imply specific constraints that will be described here.

Iterative instance set has two special properties defined as:

- A function to evaluate if a new iteration is needed or not. This function will be called each time an iteration is finished.
- A subset of break activities belonging to the iterative instance set. A break activity incarnates a new iteration's control type. Without break activities, the termination condition is checked when all the activities of the set are terminated (implicit termination). When a break activity is finished it forces the checking of the iteration condition even when other activities are executing or still not started. if the condition allows a new iteration, a new instance of each activity of the set is created and the new iterative instance set of activities can start. The new set of activities and the old one continue their executions independently of each other. Break activities allow a kind of overlapping in iteration levels.

The only structural constraints for such a set is that there should not be a path starting from an activity of the set, going to an activity out of the set and going back to an activity of the set.

Life Cycle of Multiple Instance Sets. The life cycle of multiple instance sets is illustrated in figure 4. The state diagram depends on two conditions:

Fig. 4. State diagram of a set

- **Activating condition:** The state of the set is turned on ACTIVE when at least one of its activities is ready to start.
- **Finishing condition for parallel instance sets:** The parallel instance set state is turned on FINISHED when all of its activities are finished or cancelled.
- **Finishing condition for iterative instance sets:** There is two ways for an iterative instance set to become in FINISHED state:
  - The first is when all of its activities are finished or cancelled.
  - The second is when there is at least one activity in the set that is a break activity that becomes FINISHED.

Execution of a Parallel Instance Set. As soon as an activity of the parallel instance set is ready to execute, the set is activated. This means that the number of instances is computed and that activities of the set are copied to get as many instance as needed by this result. Thus the process evolves dynamically. New activities are created, new edge are created between activities. Activities created when activating a parallel instance set are exactly the same as the activities belonging to the set except that they are marked as being clones and that they don't belong to their original set.

Edges going out of the set are also replicated. They have different origin and leads to the destination node of the original edge. This node will be a merge

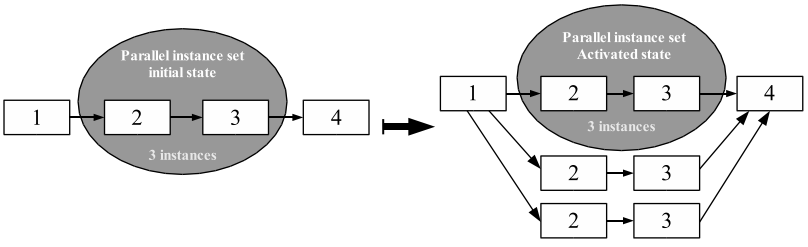

Fig. 5. A parallel instance set execution

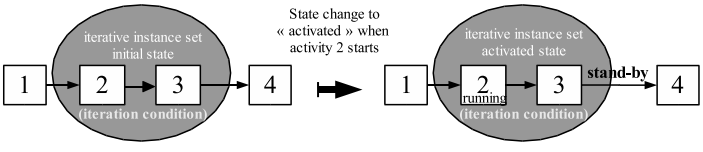

Fig. 6. Iterative instance set activation

point for the copies of the set so that the job of merging data (the different versions of the same work) is processed by this node. Figure 5 illustrates an example of activation of a parallel instance set.

Execution of an Iterative Instance Set. The iterative instance set is activated when one of the activities of the set is activated. All edges going outside the set are turned standby as described in figure 6. The standby state of an edge makes this one blocked to that state while it is not unblocked by turning it to standard state. While an edge is in standby state, activities following it cannot start.

Then, activities of the set are executed following the usual execution strategy. Once, the last activity of the set or a break activity is terminated, the iteration condition of the set is evaluated. If true, the set is re-instantiated as described in figure 7. The new copy of the set is declared as iterative instance set with the same iteration condition. The edges going outside the original set are turned to standard state so that they don't block the execution of following activities. But new copies of these edges are turned to standby state accordingly to the activation of the new iterative instance set. These new edges follow the blocking of execution of following activities.

At the final iteration, the outgoing edges are turned to standard state and there is no new instances of these edges. All the outgoing edges of all the instances of the set are in standard state and the activities that follow the set can start their execution.

4.3 Relationships between Sets

Sets can be nested or even overlap. The activation and replication process is straightforward. The rules for activation of the set remain unchanged. This is due to the dynamism of the model when any activity or set exists by itself.

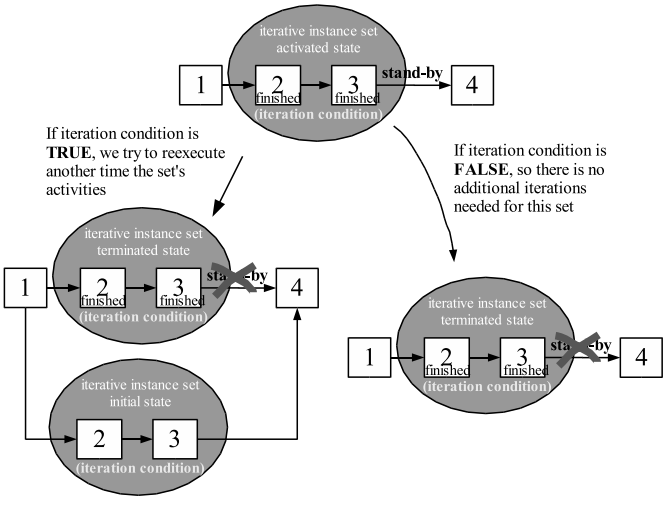

Fig. 7. Iterative instance set finishing

When activating a parallel instance set that contains other sets (parallel or iterative), these ones are also duplicated following the same scheme of cloning. A copy of a set contains the copy of activities of the original set.

When activating a parallel instance set that overlaps other sets (parallel or iterative), for each activity of the set that is also part of another overlapped set, the copies of this activity are also part of the same overlapped set (no need to copy the overlapped sets).

Figure 8 illustrate the case where a parallel instance set containing other sets and also overlapping other sets is activated.

The same approach is considered when finishing an iterative instance set that nest or overlap other sets.

Priority in Case of Concurrent Activations. Suppose the case of figure 9 where a parallel instance set and an iterative one overlap. If activity 1 is ready to start, we must decide which set will be activated first. If the parallel one is activated first, the iterative set will grow by adding the new instances of activity 1. After that the iterations will concern 4 activities. On another side, if the iterative set is activated first, there will exist an edge between the instances of activity 1 and activity 2 that is in standby state. This will paralyse the parallel instance set that will not be able to finish while iterations are not finished.

These two possibilities define an ambiguity that must be resolved by a rule: When there is two possible set activations at the same time concerning parallel and iterative instance sets, we assume that the parallel one is activated first. In the case of sets that are just parallel there is no problem of priority because the final effect is the same while activating at any arrangement.

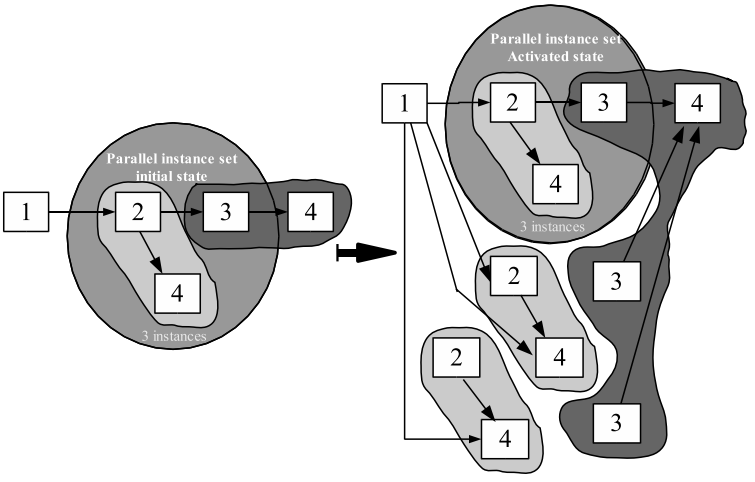

Fig. 8. Activation of a parallel instance set with nested and overlapped sets

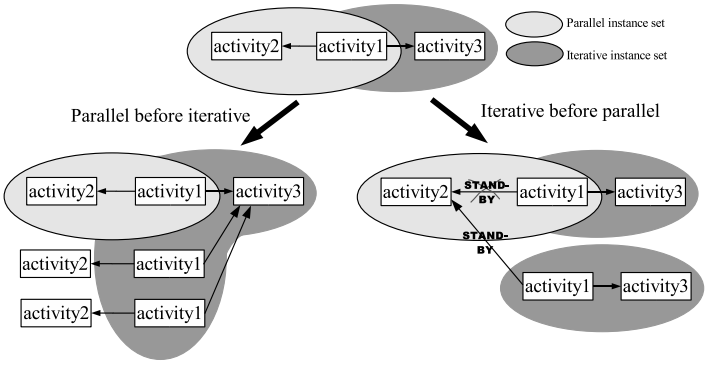

Fig. 9. Ambiguity in concurrent overlapped heterogeneous sets activations

5 Multiple Instance Sets and Workflow Patterns

The entire workflow patterns that we have presented in the introduction of this paper can be easily described as multiple instance sets like following:

- **Pattern 10**: The arbitrary cycles or loops pattern is simply mapped to iterative instance sets. The benefit is the simple definition of the set of activities and the iteration condition. This condition can be a logical formula concerning process objects states.
- **Pattern 11**: Implicit termination is implicitly used in the changing state system of sets. When all activities of a set are terminated or cancelled this set is finished.
- **Pattern 12**: Multiple instantiation without synchronization is the simple case of a parallel instance set. If there is an edge going out of the set, the

activity that this edge is going to is considered as a synchronization activity. To do a multiple instantiation without synchronization we simply don't add outgoing edges to the set.

- **Pattern 13**: Multiple instantiation with a priori design time knowledge can be simply mapped to parallel instance sets. We have there a generalized solution. Any set of activities can be instantiated multiple times. The only information to provide is a function able to calculate the number of instances that have to be created. In the case of design time knowledge, the function is simply a constant corresponding to the known number of instances.
- **Pattern 14**: Multiple instantiation with a priori runtime knowledge is also mapped to parallel instance sets. The function calculating the number of instances can calculate this number at activation time. This number can depends on any variable or property of the process. Another way is to authorize other activities to modify this function at runtime.
- **Pattern 15**: Multiple instantiation without a priori runtime knowledge is a special pattern that is generally not supported by classical workflow management systems. Multiple instance sets can offer a new solution. Oddly we don't use parallel instance sets but we use iterative instance set containing break activities. Figure 10 illustrates an example of using iterative instance sets to implement pattern 15. In this example, activity A calculates if the

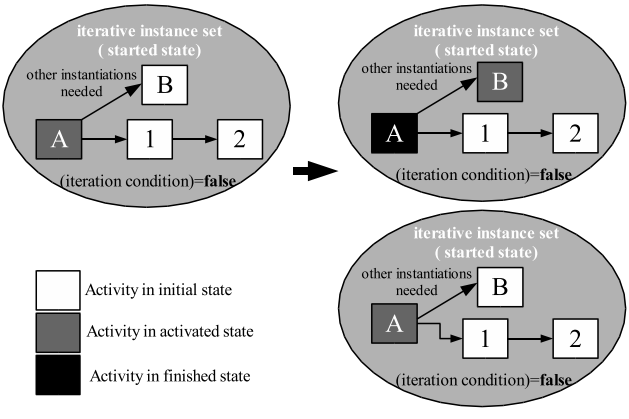

Fig. 10. The implementation of the pattern 15 based on iterative instance sets

already made instantiations (iterations) of that set are sufficient or not (instantiation time calculation). If other instantiations are needed activity B is activated. Activity B is a break activity so that when finished a new instance of the set is created. At the same time, activity 1 (the real first activity of the multiple instantiation set) is activated and the execution of the set continues normally. The next iteration that runs in parallel to the previous one applies the same mechanism until the evaluation of need of other instances is false. At the end of the execution of each copy of the set, the state of this one is

tuned on finished and the condition evaluating the need of other iterations is always null so that the finish of a set doesn't engender other iterations. This special use of iterative instance sets proves that parallel and iterative instance sets are complementary.

6 The Operette Process: Application of Set Based Multiple Instantiation

The *operette* process is an experimental process that has been executed between three classes in a school. Its goal was to produce a web site about the Opera of Nancy. This process was followed by the classes but without coordination support. This was a problem since it was difficult for each class to know what has currently been done in the other classes and what remained to be done. Adding process support to these kinds of experiments appeared as a strong requirement.

A simple process was defined at the beginning of the project, to coordinate the different classes. It describes the different steps that have to be accomplished in order to complete the project. The teachers at the very beginning of the experiment defined this process. Some of these steps had to be executed by teachers themselves, by the classes as a group or by selected group of children. This first process was designed without worry about multiple executions and is illustrated in figure 11.

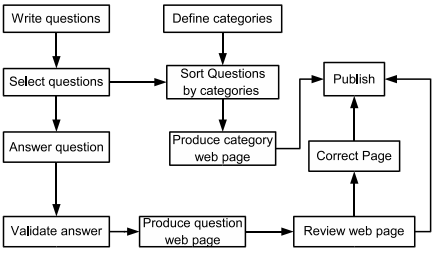

Fig. 11. The Operette process

6.1 Definition of the Operette Process Based on Multiple Instantiation Sets

The operette process can be defined using different kinds of sets, with different kinds of criteria for multiple instantiation (parallel or iterative). Some groups of activities have to be executed by a class, by a group of children, by teachers, for each question or each category. This information is not known at the beginning. Some activities have to be iterated until an acceptable state has been reached. This is the case before publishing the web site. Pages are corrected and reviewed until they are correct. Figure 12 describes the different sets that could have been defined for the process and the criteria used for multiple instantiation.

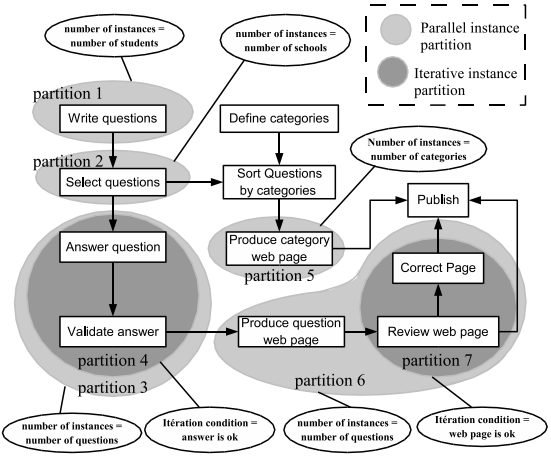

Fig. 12. The operette process with sets

Upon execution, the new activities are created and synchronized as defined by the model. We can see clearly the use of parallel instance sets with execution time knowledge of iteration number. That's the case of the set III where the number of questions is unknown at design time. Moreover, this set contains an iterative instance set. Across this example, specifying the sets is quite simple and specifying the original workflow is clearer.

7 Implementation over the BONITA Model

The implementation of our set based model has been made on the Bonita WFMS as an update to support sets. Bonita is a Dynamic Cooperative Workflow Management System implemented on a J2EE platform and is an ObjectWeb project. The Bonita workflow model is dynamic and flexible. Processes are not instantiated from a process definition but directly created by users and executed. Users can create a new process from scratch by adding activities and edges between activities. They can specify properties of activities. An activity can become executable as soon as it has been created. Process creation can also be done by reusing other existing processes. A process can be imported into a running process. In this case a copy of all the activities edges and properties is done in the target process. The user can then make the modifications he needs to adapt it. Starting a new process can be done by cloning an existing process to avoid starting the definition from scratch. As we don't expect that cooperative processes are executed so many time and that they often require adaptation to meet each project needed, we think that this way to define process avoid the complicated programming stage of process definition. Moreover, the definition of a cooperative process is often done by analogy to processes that have been executed in the past with some adaptation. This provides a very flexible way to create processes, to adapt them, to change them or combine them on the fly.

The current definition of the Bonita model provides flexible definition and flexible execution of processes. The flexible definition is achieved by allowing changes to occur at any time during execution of the process. The only and main constraint is that an activity definition cannot change once it has been started. Other constraints are structural. For instance a cycle cannot exists between activities. Flexible execution is achieved by a dynamic management of the activity state and by allowing anticipation of execution of activities. An activity may be started even if all the conditions for its execution are not fulfilled.

Scripts can be associated to activity state changes in order to automate certain parts of the process and these scripts are called Hooks. The execution of a hook can interact with the workflow process and can for example create, modify or remove other entities in the workflow. A hook can be assigned to be executed before or after the start or the end of an activity. Bonita workflow model has been implemented and integrated with different kinds of extended transaction models to allow object sharing between activities and to support concurrent long running activities.

All these properties allow the possibility to extend the bonita model to support multiple instantiation sets. The internal work of sets in Bonita is defined as EJB. The update of Bonita has been simple because the use of sets is independent of the workflow engine behaviour. The most important update to do is located on the activity state manager because the set activation is the result of an activity activation. There is two possibilities to do this change. The first is an update of the internal engine that controls activity states so that each time an activity starts, all the sets containing this activity are activated as needed in the model. The second possibility is the use of Hooks. A hook can be associated to all the activities of the process with a "before start" activation. This hook activates all the sets containing the activity as needed in the model.

We have chosen temporarily the engine update to implement that and to make tests on the sets uses but we are now integrating the solution based on hooks. The current solution uses the Jboss web application server but a newer implementation based on hooks and using the Jonas web application server is currently developed. All the files of the Bonita project are located on the ObjectWeb consortium web page (http://objectweb.org).

8 Conclusion and Perspectives

Specifying set of activities that have to be executed an unknown number of times, in sequence or in parallel is a challenge in process models. The use of control structures from traditional or parallel programming language is not adapted to the specific needs and usages of business or cooperative processes. Additional operators that have been added in other workflow models are difficult to manage and lead to complex structure to express simple things. In the Bonita model, creating an iterative or parallel instance set does not change the general intuitive structure of the process. Process execution is at the same time a mean of control, of automation and of awareness. Changing properties directly on objects or set of objects affects the behaviour of the whole system. The interface provided with the implementation allows monitoring dynamically the evolution of the process.

The solution based on sets provides a simple way to define parallel or iterative multiple instantiations. Sets of activities can be selected with practically no constraints so that we can choose dispersed activities in the workflow. In other workflow models, group of activities to multi-instantiate must be sequentially connected or simply just one activity can be instantiated multiple times. The set based workflow structure is easier to understand by the end user. The main strength of sets model is that it allows the implementation of the entire multiple instantiation patterns defined in [9]. Moreover, set based multiple instantiation allows nested and overlapped sets that is a very useful new possibility in some cases.

Now we are planning to integrate this work in a software development platform that involves data synchronization services. This platform is the LibreSource platform and our next challenge is to integrate the control flow functions of bonita and data flow facilities of the platform. At the same time we are planning to develop an advanced transaction manager probably based on the same approach.

References

1. M. Dumas and A. ter Hofstede. Uml activity diagrams as a workflow specification language. *In Proc. of the International Conference on the Unified Modeling Language (UML). Toronto, Canada, October 2001. Springer Verlag.*, 2001.
2. D. Grigori. *Eléments de flexibilité des systèmes de workflow pour la définition et l'exécution de procédés coopératifs.* PhD thesis, Université Henri Poincaré - Nancy1, Ecole doctorale IAEM Lorraine, Paris, novembre 2001.
3. Daniela Grigori, François Charoy, and Claude Godart. Anticipation to enhance flexibility of workflow execution. In *DEXA 2001*, number 2113, pages 264–273. LNCS, 2001.
4. Manfred Reichert and Peter Dadam. A framework for dynamic changes in workflow management systems. In *DEXA Workshop*, pages 42–48, 1997.
5. W.M.P. van der Aalst. How to handle dynamic change and capture management information, 1999.
6. W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1–2):125–203, 2002.
7. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced workflow patterns. In O. Etzion en P. Scheuermann, editor, *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer-Verlag, Berlin, 2000.
8. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *BETA Working Paper Series, WP 47, Eindhoven University of Technology, Eindhoven*, 2000.
9. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):pages 5–51, 2003.
10. Jiantao Zhou, Meilin Shi, and Xinming Ye. On pattern-based modeling for multiple instances of activities in workflows. *In International Workshop on Grid and Cooperative Computing, Hainan*, pages pages 723–736, December 2002.