A Dynamic Workflow Management System for Coordination of Cooperative Activities

François Charoy, Adnene Guabtni, Miguel Valdes Faura

A Dynamic Workflow Management System for Coordination of Cooperative Activities

François Charoy(1), Adnene Guabtni(1), Miguel Valdes Faura(2)

(1) University Henri Poincaré Nancy 1 - INRIA - LORIA laboratory,
BP 239, F-54506
Vandoeuvre-lès-Nancy Cedex, France
(2) Bull R&D, 1, rue de Provence
38130 Echirolles (France)
Francois.Charoy@loria.fr

Abstract. This paper comes back to the problem of coordination of cooperative activities with a Workflow management system. First, we describe the differences that we have noted between business processes and cooperative processes. Then we present a set of requirements for a Workflow management system that aims to support cooperative workflow, and among these requirements are high flexibility and dynamicity. Then we describe how this has been taken into account in the development of the Bonita workflow management system that proposes to remove the idea of process model to work only with process instances that can be derived from each others or that can be composed.
Keywords:Adaptive processes,Cooperative processes,Architectures and tools for dynamic processes.

1 Introduction

Using workflow technology to support cooperative activities is an old idea, taking its sources in Office Information Systems. A lot of work has been devoted to this problem during the 90's with the advent of the CSCW field. It must be noted that although automation of business process management and web services composition has gained in visibility and acceptance, its application to coordinate cooperative work is not yet a success. But with the greater acculturation of people to cooperative work over the Internet, the need for better support for coordination if beginning to appear with a greater pressure. A lot of domain, such as e-learning, software development, content management systems, scientific and medical applications, crisis mitigation systems require now better support for coordination and tracking of individual activities.

One of the assumption that has been made some years ago is that workflow and business process modeling could be used, regarding some evolutions, to support the coordination of cooperative activities. A common belief is that the ability to easily change process types or process instances is still considered as an important issue for acceptance of Workflow management System in organisations. In a cooperative environment, this requirement is even more important.

Business processes can be considered as stable regarding cooperative processes. A business process takes time to be designed and implemented, but this cost is redeemed by the number of its execution and by the expected raise of productivity. Cooperative processes like software development processes are of different nature. They are long lasting processes with a high potential for evolution during their life-time. They are not executed so often. Spending a lot of time to design and implement such a process would be considered as a waste of time.

We consider, following the proponents of ad-hoc workflow for cooperative processes that, defining a complete cooperative process from the beginning with all its details is almost impossible. The process for software development or for technical report production may not be known entirely at their beginning. They may have to be refined during their execution. Our point of view is that a cooperative process will evolve during its execution. Thus, it must be very easy to change during its execution, instance by instance. We try to push this hypothesis to its extreme by not making the difference between model and instance.

To summarize, a Workflow Management System that aims to support cooperative activities must first provide the same kind of support as a WFMS for Business processes i.e. activities, activity dependency management, performer and resources management. It must also be very flexible and allow easy modifications by the users, instance by instance.

In the first part of this paper, we will try to summarize the differences between so-called business processes and cooperative processes. Then we will list the requirements that we want to met. The next part will present the model underlying the Bonita Workflow Management System and how it's flexibility can potentially meet the requirements that we have described.

2 Business vs Cooperative processes

Cooperative and business processes are different in nature. We can identify a number of differences between a WFMS that has to support cooperative process and a WFMS that has to support Business Processes. These differences are the following:

The number of process models In a cooperative environment, the ratio between the number of execution of a process and the number of process definitions is small compared to a business environment. Processes are built from process fragments on a project by project basis. Pushing this assumption to the extreme, all processes in a cooperative environment are different. That means essentially that the participants to a process must be able to design the process.

The process structure is simpler Business process can be relatively complex with many alternatives, compensating activities and the rest. When defining a business process, designer try to consider almost every possible case. Cooperative processes are simpler in general, consisting on sequences of activities executing

in loops[1]. We consider that cooperative process are generally the concatenation of successive steps that lead to the production of a final result for the project. It may still happen to find a complex structure inside the steps. However, the organisation of the process itself must be understandable by the participants to the process. Thus it cannot be too complicated.

The process evolves more often A business process is less subject to evolution than a cooperative process. The long duration of a cooperative process encompass in itself the need for change. Changes in the environment or in the goal of the process have more chances to occur. A business process is supposedly shorter and thus less subject to changes during its execution time.

The process is user driven Cooperative processes governs cooperative projects where participants are concerned by a common goal. This is less the case in a classical business process where people are mostly concerned by the task they have to execute. Thus, cooperative workflow management systems should provide their users with a clear view of what has been done, who does what and what remains to be done (even if that is expected to change). A cooperative process is the result of a consensus between its participants.

Althoug these differences are important, we still think that it is possible to adapt workflow models and workflow management systems. in the next part we describe the requirements that are important for such a system.

3 Requirements for cooperative processes support system

A system that aims to support explicit cooperative work coordination needs to provide some incentive to its users. Even if there is a feeling for better support for coordination in distributed cooperative project, and even if there is some will to set up clear procedures at the beginning, the use of these procedures and tools to track the activities is most often forgotten as soon as the project begins and the pressure to get results raises. If the users lose the feeling that following the process is useful for them, they stop using it, or use it lazily[1, 2].

Designing the processes In a cooperative setting, the plan and the process to follow is generally the result of some consensus after discussions between the members of a team. Decisions are taken and actions to be done are distributed among them. Most of the time, these actions are small processes that have to be refined and connected to the overall group coordination.

For instance, when doing software development, the development plan is decided and then refined. Creating a cooperative process from scratch would be very long and error prone. Most of the time, when starting a cooperative process, even with implicit coordination, people refer to existing process that they have executed before. They try to reuse part of their previous experience in the new

[1] most design process for instance require several execution of the same activities to reach a given result

project. A cooperative workflow management system must be able to reproduce this behavior by allowing the reuse of fragment of processes that have already been executed.

A cooperative WFMS must provide an efficient library of process instances or fragments that can be easily integrated in a new process (for instance a process for paper reviewing or a process for code release in a software development project)

Users control the process Users participating in a cooperative process should be able to monitor and change the process. Business process are constrained by management and cannot be changed by end-users. This is the contrary in a cooperative process. We consider that the cooperative process is the result of its execution (the process is itself a product of the process). Each user has the ability to add/remove/change activities of the process in which they participate.

Automation of activities The incentive to use a workflow management system is not always clear for user. A lot of experiences have shown that if the users find no benefit using this kind of system they will avoid to use it by any means(see [3] and related works). A cooperative workflow management system should provide some assistance, and some automation for the most repetitive tasks and not just control for the management. It must also be sufficiently integrated to the environment to help the users to find the documentation they need to modify, to publish, retrieve and share these documents, to track what happens to them. If these conditions are fulfilled, there is more chance that users will contribute to the process evolution[2].

In order to reach these goals we have started a project some years ago that has resulted in the development of a Workflow engine called Bonita that has been already used in different settings. The Bonita model was designed with these requirements in mind.

4 The Bonita Model

The definition of the model has been done with several constraints in mind. Current standard process models are complex. The definition of a process requires specific skills. They are reserved to specialist and cannot be read by common users. Although we know that the complexity of business process definition may require some expertize, we argue as we have said before that cooperative process need to be managed by their end users.

The definition of the Bonita model is inspired from dynamic languages [4]. It does not separate the model (a class) from its instances (objects). Instances can be directly created from the Bonita API, executed and modified dynamically. A new process can be created either from scratch, by cloning an existing process or by importing a process definition inside an other process. On figure 1, on the left is a window showing the state of the process for a user, and on the right is

[2] Of course this remains to be proved by experiences

the state of the process. This window is also an editor, thus the process can be changed at any time.

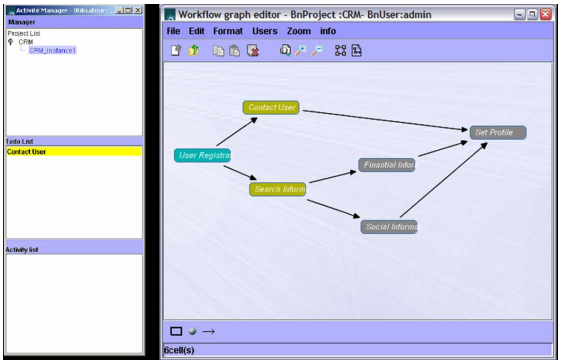

Fig. 1. The execution and definition interface

A process describes a set of activities that have to be executed to reach some goal. It has participants that can adopt roles within the process. These roles are used to create the relationship between activities and the user that can execute them. Constraints on the definition of a process are kept minimal to ease its definition by end users. A process is a defined by a set of activities and by dependencies between activities. Dependencies between activities are end/start dependencies, with join conditions and split conditions.

A process is created by a user, the owner of the process. From this point, the process is started. The owner can add activities to the process and dependencies between these activities. A process is started by its owner. It can be terminated automatically when all activities of the process have been terminated, aborted or are dead. It can also be terminated or aborted by a user explicitely.

Activity states are the following : initial, executable, executing, anticipable, anticipating cancelled, aborted, terminated. An activity can be executed as soon as it is created. It is then in the state Executable.

Flexible execution of processes is possible due to the ability to start an activity in advance. This is what we call anticipation. Anticipation which has been already described[5] is a mean to reduce constraints on the execution of cooperative activities. The main idea is that an activity can be executed even when all its activation conditions are not met. But we guarantee that at some time before its termination, they will be met. Thus, at the end of the process execution everything appears to have been executed normally even though some activities have been started before there normal activitation time. The main advantage is that even with strict process definition, flexible execution remains possible. Figure 2 is a case where node1 has been started, node2 and node3 have been started with anticipation and node4 is in a state where it could be started.

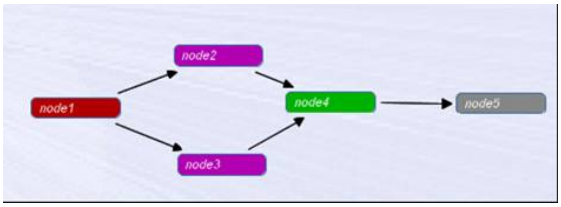

Fig. 2. Some activities can be started even when the preceeding ones are not finished

New activities and new dependencies can always be created during the process execution. The only constraints concern the state of activities. It is not possible to change activities and dependencies concerning terminated activities.

The owner of a process can also attach users to the process with specific roles. Users are then participating to the process. They belong to different roles. An activity can only be executed by a user that can take the role specified for the activity.

The workflow engine is able to calculate to do list and executing list for each user participating to a process and to notify users of every change that concerns them.

4.1 Process Building blocks

Of course, defining each new process activity by activity is not a very sound way of working even though we are doing that very often in real life project. The support of the WFMS must appear as valuable and in this case, the risk is that some activities are created at the beginning of the project and no followup occurs. This is often the case with planning tools and can be verified in many open source projects on Sourceforge for instance : a project is created, many tasks and activities are instanciated and assigned and then nothing more happen.

In a cooperative project, the process must be described very easily, based on previous experience. Writing a document as a group has been done many time. If their process require such kind of step, a group of users must be able to find several process fragments that provide a solution for that (plan, edit, review, release or plan, produce, edit, release).

The following example shows different steps in the life of a cooperative project.

Fig. 3. The initial process

On figure 3, a small editing process is used to start the production of a document. Several activities are instantiated. Then the process is started and a validation/submit step are added to the process (figure 4).

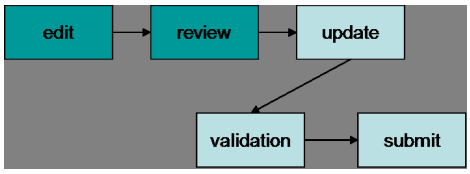

Fig. 4. The process is completed with validation/submission

From this result a new document has to be produced. Thus, the editing process is imported again and connected to the validation activity (figure 5).

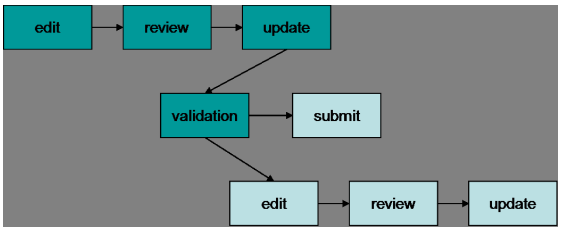

Fig. 5. The edition process is reused for a new document

Process importation and Process cloning are the two main mechanisms that we propose to support this kind of behavior. We follow the path of prototype based language that do not make the difference between classes and instances. Any process instance, running or terminated can be used to instantiate a new process. In this case, activities are reinitialised to their initial state and every properties of the original process are imported in the new one, except users. Thus a process can be build by importing different processes and then by creating dependencies between activities of these fragments. A process can be suspended during this phase. If it is not, state of activities is immediately updated to reflect the new state.

The dynamicity of the model allows this kind of behavior. Dependencies can then be created between existing activities of the process and the imported ones.

4.2 Data Flow

A process is not just about coordinating activities. It is also about managing the data that are used by these activities. Our model provides some simple support

for process data and has been integrated in a more sophisticated environment for shared data management.

Two kinds of data are directly managed inside a process. Process data and activity data. Process data are properties that can be access and changed by all the activities of a process. Activity data acts as input and output parameters. Each activity has a list of input and output data. These data are represented as properties, with a name, a value and a read/write constraint. Then these data are propagated to the succeeding activities. Conflict may occurs when two properties with the same name are propagated to an activity through an and join node. In this case, we choose to keep the last value for simplcity reasons, but this point need to be consolidated. In cooperative activities, we consider that activities use mostly data from shared workspace (document spaces, source repository) where they commit and checkout data when they need it. For these data, we consider that to each activity, a local workspace is created where the shared data are checked out at the beginning of the activity, and checked in at the end of the activity. Thus conflicts and concurrency problems are managed by the shared repository and depends on its protocol.

4.3 Process Correctness

We put very few constraints on the structure of the executing process. This is the cost of dynamicity. Only cycles are detected and forbidden except when they belong to the special iterator construct. A process is always valid. Activities are executable as soon as they meet their start or anticipation condition. A process is considered as terminated when all its activities are dead (not reachable) or terminated. This is a very different approach than the ones that are generally considered in business process management, but we think that flexibility is more important than consistency in this context. As the process is not supposed to be executed a great number of time, consistency problems can be solved when they occur.

4.4 Automating activities

Acceptability of a process control by users depends on the benefits that the users can obtain from the process execution itself. Automation of part of the process is one of these expected benefits. Although many activities in a cooperative workflow are user driven, there are still large part of them that can be automated. Test, compilation, and different kind of supports that can be implemented by services provided by the process execution environment.

In our model, we allow the attachment of scripts that we call hooks to state changes of activities. For instance a script can be associated to the state change from *executable* to *executing* or from *executing* to *terminated* of an activity. When several hooks are associated to the same state change of the same activity, they are all executed in an undefined order. For instance, when a user has finished and editing activity, its workspace can be automatically checked in in a shared repository.

Special kind of activities can be defined as completely automatic. As soon as they become executable, they are executed and all the scripts associated to their state change.

Failure of the execution of a Hook cancel the state change. Thus, hooks can be used to express termination condition on activities. For instance, they can be used to check the status of an activity when the user tries to terminate it. If the check fails, the activity remains in the executing state. Note that state change of activities are atomic and include hook execution. Hook implementation is done in Java or with a script language (BeanShell). Conditions can be expressed using the support provided by the language.

Hooks can be specific to an activity or associated to the process. A hook associated to the process will be executed for a specific event for all activities of the process. This allows to adapt the general behavior of a process.

Of course, hooks can make the definition of a process complex as it requires some programming. Our goal is to provide library of hooks for very generic actions and to provide the ability of using script language to describe simple action. Hook have access to the context of the current activity and they can be used to call WebServices in the scope of the activity execution transaction. Figure 6 is an example of such a hook definition in Java that sends an email when the correct state is reached. This hook is associated with an activity and is executed when the activity is started. The parameters of the hook are objects containing the context of execution, i.e. the activity and the process data.

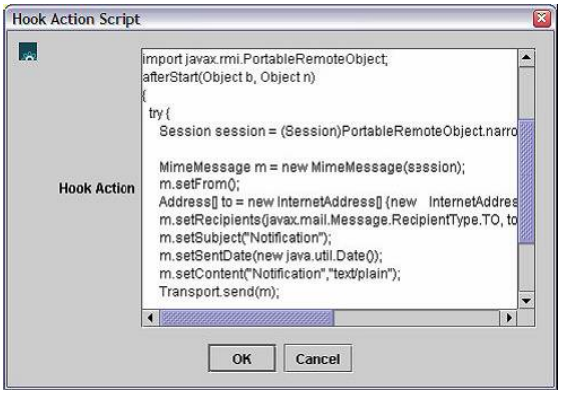

Fig. 6. AfterStart hook implementation

Hooks can also be used to modify the current process. We plan to use hooks to generate compensation process when an activity is cancelled or aborted but this is still an ongoing research.

4.5 Role management

Role management is classical. To each process is associated a set of role and activities are associated to roles. User can take role and they can execute activities that are associated to one of their role. Procedure (performer assignements) can also be attached to activities to calculate user assignement.

4.6 Awareness

Every event (process change or state change) on a process produces an event that is published in a message queue. Users may register to be notified of these events. They can choose to be notified of every activity termination for a given process. They will receive an email or an instant message. This is a very basic form of awareness. The process edition tool is also kept synchronized with the current state of the process. This is interesting but no so useful as we consider that the pace of execution of a cooperative process is relatively slow, so events will not occur so often.

5 The implementation

The Bonita System (bonita.objectweb.org) is available as Open Source and is actually in use but more for classical business process management than real cooperative one. Its development has started in the LORIA lab and the main support is now provided by Bull R&D. It is implemented on a J2EE Jonas Server and uses the Jabber XMPP protocol for event notifications. Rich Swing clients are maintained up to date with JMS events. A Web interface is also available for an access behind a firewall. Figure 7 provides a view of the Bonita architecture.

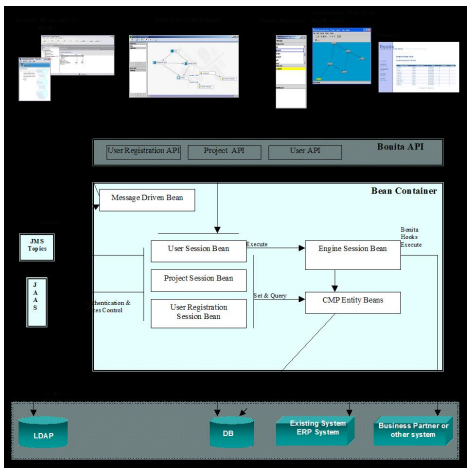

Fig. 7. Architecture of Bonita

6 Related Work

A lot of work has been devoted to the problem of providing a dynamic of flexible process environment. Some work was devoted to the management of change in processes [6], change in process definition through different techniques [7]. Flexibility and exception handling has also been proposed to manage unexpected situations [8, 9]. Other approach like [10] allows for dynamic changes to the process instance but restrictions on the operations that can be applied in order to maintain some consistency. More recent work [11] proposes to combine a classical workflow model with some pocket of flexibility that reduce the constraints on execution. In [12], the authors uses the idea of emergent workflows to allow adaptation of process instances at runtime. It combines planification and workflow management.

A general study on state of the art of correctness criteria for dynamic change in workflow can be found in [13]. The goal of most of this work is to maintain the consistency between the process model and its instances in case of instance or process evolution. All these approaches provide interesting insights on the different kind of flexibility while keeping a correct workflow structure. Our point is that structural consistency is not as important as the ability to build easily dynamic processes that can be controlled by users.

Other works take different directions that are not based on Workflow Management systems. Some years ago, we tried to control process using temporal constraints [14]. In this work, the process was not defined but the state of the system was driven by constraints that forced the system state to go through different stages. The results were interesting but the constraints were difficult to write and to understand for end users. Other work have done in the same direction [15]. Our point of view here is that although rule based systems or constraint based system are interesting, they fail to provide the correct level of support to end users and are hard to maintain. This is why, even with its limitation, the workflow approach is still the best one for us at this time.

7 Conclusion and Future work

With our approach, we have pushed to the extreme the idea of flexibility in workflow management. The process execution is considered as a program execution where the program is written at runtime by its users. Processes are created by hand or by importing or cloning existing processes. Of course, this limit the kind of consistency control that can be done on the process structure, but this allow also quick corrections in case of problems. The only part of the process really known is the one that has already been executed. This requires also to support process definition with library of predefined process fragments that solve generic problems that may occur in cooperative processes (some kind of cooperative process patterns) that need to be defined. It means also that we need to provide more help for users with for instance a greater integration of the process with the user environment. At best the WFMS should be able to guess that the

user is working on a given task and even that he has finished to work on it. It should also provide some kind of ubiquitous todo list management system, easily accessible by users. These are some paths that we plan to explore in a near future.

References

1. Charoy, F., Godart, C., Gregori, N., Hautecouverture, J.C., Jourdain, S.: Coopera : An environment for teaching and learning internet cooperation. In: IADIS International Conference e-Society 2004, Avila, Espagne. (2004) 323–330
2. Herrmann, T., Hoffmann, M.: The metamorphoses of workflow projects in their early stages. Computer Supported Cooperative Work (CSCW) **14**(5) (2005) 399 – 432
3. Suchman, L.A.: Plans and Situated Actions : The Problem of Human-Machine Communication (Learning in Doing: Social, Cognitive & Computational Perspectives). Cambridge University Press (1987)
4. Ungar, D., Chambers, C., Chang, B.W., Holzle, U.: Organizing programs without classes. Lisp and Symbolic Computation **4**(3) (1991)
5. Grigori, D., Charoy, F., Godart, C.: Coo-flow: a process technology to support cooperative processes. International Journal of Software Engineering and Knowledge Engineering - IJSEKE Journal **14(1)** (2004)
6. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: COCS '95: Proceedings of conference on Organizational computing systems, New York, NY, USA, ACM Press (1995) 10–21
7. Joeris, G., Herzog, O.: Managing evolving workflow specifications. In: Conference on Cooperative Information Systems. (1998) 310–321
8. Hagen, C., Alonso, G.: Flexible exception handling in the OPERA process support system. In: International Conference on Distributed Computing Systems. (1998) 526–533
9. Luo, Z., Sheth, A.P., Kochut, K., Miller, J.A.: Exception handling in workflow systems. Applied Intelligence **13**(2) (2000) 125–147
10. Reichert, M., Dadam, P.: ADEPT flex -supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems **10**(2) (1998) 93–129
11. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. Inf. Syst. **30**(5) (2005) 349–378
12. Bassil, S., Keller, R.K., Kropf, P.G.: A workflow-oriented system architecture for the management of container transportation. In Desel, J., Pernici, B., Weske, M., eds.: Business Process Management. Volume 3080 of Lecture Notes in Computer Science., Springer (2004) 116–131
13. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems–a survey. Data & Knowledge Engineering **50**(1) (2004) 9–34
14. Skaf, H., Charoy, F., Godart, C.: A hybrid approach to maintain consistency of cooperative software development activities (1997)
15. Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., Zbyslaw, A.: Freeflow: mediating between representation and action in workflow systems. In: CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work, New York, NY, USA, ACM Press (1996) 190–198