Constraint-based flexible workflows

Jacques Wainer and Fábio de Lima Bezerra

IC - UNICAMP
Avenida Albert Einstein, 1251
Caixa Postal 6176, CEP 13083–970, Campinas, SP - Brasil.
{wainer,fabio.bezerra}@ic.unicamp.br

Abstract. This work presents the idea and a prototype of workflow systems defined through constraints. We think that in many domain areas, such as health care, workflow systems are too inflexible. One does not know in advance what activities does the patient has to go through. Actually a new form of interaction with the workflow is needed, in which a controller asks the systems which precondition activities are necessary in order to execute a target activity and schedule the target activity. We also present Tucupi, a prototype of such constraint based WFMS.

1 Introduction

Workflow management systems (or WFMS) are systems that allow for the specification, execution, and monitoring of business processes. In execution, an instance of a process, or a **case** goes through a series of **activities** performed either automatically or by people, according to a pre-specified **process definition**. In usual business domains, the process definition is created well in advance of its use, and although most process definition formalisms allow for process that have some flexibility, this flexibility is defined in advance as conditional execution of different (pre-defined) paths based on some of the case data.

In particular, we are interested in the domain of health care. In this domain, as well as others, there is the need for what we will define as **partial workflows**, that is, workflows that are only partially defined and are conditional. When a patient is admitted to an hospital, one does not have information to define, at that time, all the activities will be performed to/one behalf of the patient. Clearly, after the patient is discharged, one can look back and see the patient's stay as the execution of a workflow: activities ordered in time where performed to/on behalf of the patient, but at no moment there was a workflow specification that of which the patient's case was an instance.

So patient care, as many other activities, such as software design, are a kind of **plan as one goes along** workflow. The execution of the case, or better the particularities of the case as it unfolds, defines which activities should be performed next, or in the future. But whoever is in control of the workflow case, does not have total freedom to decide which activity must be performed next. There are rules or constraints that must be followed that imposes the execution of unrequested activities, that forbids some activities for being executed or at least being executed now, and delays the execution of activities, and so on.

Moreover, in domains with less defined processes, usually there is a person, or people, which we will call the **controller** who decides which activities must be performed to the case, which we call the **target** activity. In the hospital example, a physician (the controller) decides that a surgical intervention is needed (the target activity).

In the workflow as a helper scenario, the workflow will help the controller decide on the implications of the target activity. The implications certainly involves the need to execute activities both before and after the target activity (which we call **forced** activities). In some domains, temporal constraints regarding the forced activities may place further constraints in the execution of the target activity. For example, an organization may define a rule that meetings must be proceeded by the distribution of the meeting preparatory material at least two days before the meeting itself. Thus the workflow as a helper can inform the controller that to call a meeting with the development team, he must generate the preparatory material, and if he does that immediately the sooner the meeting can happen is in two days.

However, standard workflow systems are inadequate to model and enact such work activities situations. Workflow systems, as used in the business environment, only allow for well defined processes. The workflow definition for such processes is total and complete: the workflow defines all the activities that will be executed in a case and in which order. Once activity A has finished the system can compute exactly which activities must start now. In such defined processes, the workflow system works as a dispatcher – at the end of an activity it computes which activities become enabled and dispatch the case to the executors of such activities. The control of what will be executed is in the WFMS, and some control of when the activity will start (once enabled) is left to the executors themselves (if the WFMS uses the concept of inboxes).

This paper is organized as follows: in section 2 we present the representation of the constraints in the workflow definition; in section 3 we present the Tucupi, a prototype of a workflow system that it uses workflows based on constraints; in section 4 a discusion about temporal reasoning is presented; in section 5 we present some related works and in the section 6 some conclusions and suggestions of extension that can be implemented.

2 Constraint representation

This section will present the form as the restrictions that the definition of a process composes must be represented. For in such a way, we will use a restricted language. The language define both the preconditions and post conditions of a target activity. The simplified version of a constraint is illustrated in the following example:

We call such rules above as **order constraints**. The order constraint `c1` specify that to perform the target activity A, activity B must have ended (`precondition:`

```
rule: c1
    target:  A
    precondition:  B
    precondition: C
    postcondition:  D
```

B), activity C must also have ended, and activity D must happen after A is ended (`postcondition: D`). The full complexity of the constraint language is reached by adding the following constraints constructs:

– the `precondition` can be a disjunction of activities

  ```
  target: A
  precondition:  B or C
  ```

  state that in order for A to be enabled, either B must have happened or C must have happened;
– also in `precondition` there can be a constraint that an activity should *not* have happen before the target activity

  ```
  target: A
  precondition: not b
  ```

  state that in order for A to be enabled, B must not have happen before;
– the construct `parcondition` (after parallel condition) states that an activity must happen either before or after the execution of the target activity

  ```
  target: A
  parcondition: B
  ```

  state that if A is executed then B must be also executed.

The constraints also can be used to represent a traditional (or total) workflow defined with the structures: AND-Split, XOR-Split, AND-Join, XOR-Join and sequencing. Space constraints do not allow us to elaborate this further.

3 Implementation - The Tucupi Server

The core of the system is the Tucupi server, which at execution time answers queries, and accept commands and updates from the controller. In fact, the server receives queries and commands from a known source but does not verify that the source is a or one of the authorized controllers. It assumes that some other software component will perform the authentication.

The server answers to the following queries:

who(A,C,W) This query returns the users that can perform activity A for the case C of the process W. This is a pure WRBAC query;

enabled(C,W) This query returns the list of enabled activities for case C of process W. An activity is enabled if its preconditions were satisfied;

not-ready(C,W) This query returns the list of activities that are not yet enabled for the case C of process W. This includes forced and target activities which do not have all their preconditions satisfied;

what-if(A,C,W) This query returns a chain of precondition activities of all forced activities that have to be executed in order to perform the target activity A. For example, if an activity A1 has activity B1 as a precondition activity, and B1 has activity C1 as a precondition activity, then the query will return activities B1 and C1.

The server also accept the following updates and commands:

started(U,A,C,W) This command informs the server that activity A for case C of process W started now and that U is the executor of that activity ;

ended(A,C,W) This command informs that activity A for case C of process W ended at this time;

add(A,C,W) This command informs the server to add the target activity in the future of the case C for process W. The corresponding forced activities are either placed in the enabled list (if they have no precondition themselves) or in the not-ready list.

3.1 Architecture

The architecture of server is composed of three components as depicted if figure 1:

WRBAC Component . The component that implements the WRBAC model [7]. WRBAC is an extension of the Role-Based Access Control (RBAC [8, 1]) mechanism to workflows. WRBAC adds to the RBAC concepts that of a *workflow case*, and associates to each activity in a workflow a corresponding privilege to execute that activity. Using the concept of a workflow case it is possible to model *dynamic* rights and non-rights in extension to the RBAC *static* modeling of rights. Thus, WRBAC Component is used as a mechanism of access control.

Workflow Definition Component . The component used by the user to define the process using the rules presented above.

Workflow Engine Component . It is the main component. It is used to process the queries made by users. Moreover, it is used as a mechanism of execution, for example, when the user starts (or ends) a workflow or activity.

The server was implemented in Prolog, and the current version achieve persistence of the dynamic information by saving the state to a file after each new update. This, of course, places some limits on the frequency of updates the system can handle, but such limit was not reached in the tests performed (a few updates per minute).

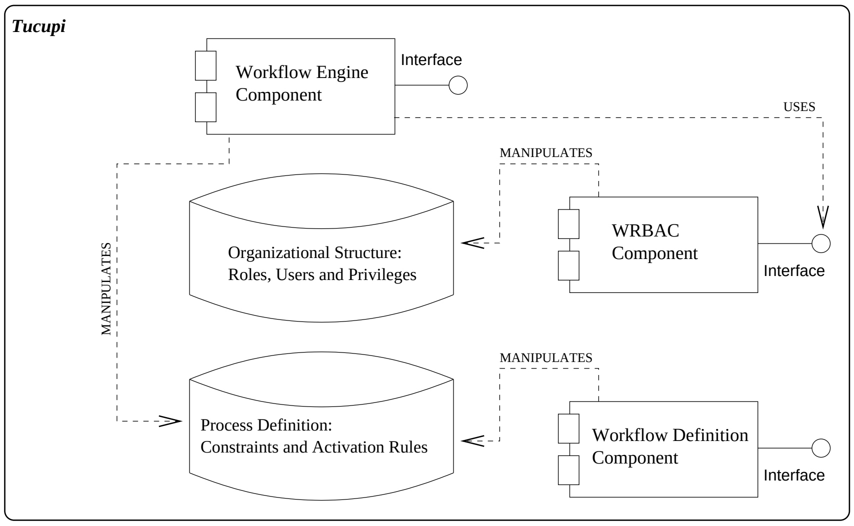
Tucupi
Workflow Engine Component
Interface
USES
MANIPULATES
Organizational Structure: Roles, Users and Privileges
WRBAC Component
Interface
MANIPULATES
MANIPULATES
Process Definition: Constraints and Activation Rules
Workflow Definition Component
Interface

Fig. 1. The Tucupi Architecture

3.2 Rule derivation

The representation of workflow as a set of restrictions has limitations. The definition of workflow needs to include declarations that indicate when and which activity must be executed during the execution of a case. However, constraints are declarations that indicate what is not allowed to be made, what [9] called negative rules. The positive rules are rules that indicate what it must be done, therefore the opposite of the negative rules.

Rule derivation is a transformation of the rule as presented above into two new rules: a consistency rule and an activation rule. The consistency rule maintain the original semantics of the constraint, whereas the activation is a positive rule.

The figure 2 illustrates an example of rule derivation. The derived rules had been written in Prolog. The predicates **ended** and **started** are update commands presented above.

A single rule may generate more than two (Prolog) constraints, depending on the granularity needed for the internal representation of the constraints. This is irrelevant for the current work, but as we discuss in the last section, for future work we expect to add the “constraint overriding” feature of WRBAC [7] into this system, and in that case, exactly how much of the rule is embedded into the constraints is important, since it is these internal constraints that will be overridden.

In the example presented in figure 2, we have a constraint $C01$ generated from rule with the same name, $C01$. The derivation states that $C01$ will be a violation if b try to start and a has not ended yet. The activation rules are generated for each postcondition and for each target activity. These rules dispatch the

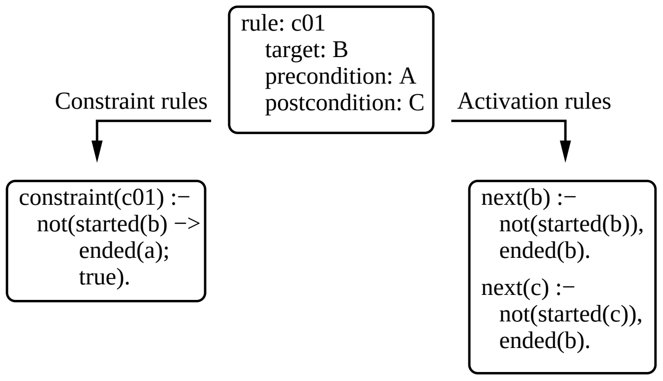

Fig. 2. A rule derivation example

activation of activity indicated in `next` function argument. In example presented in figure 2, we have a `next(b)` function generated from target activity B and a `next(c)` function generated from postcondition C. The `next(b)` function states that b will be the next activity if b has not started yet and the activity a has ended. The `next(c)` function states that c will be the next activity if c has not started yet and the activity b has ended.

4 Temporal Reasoning

Finally, both the representation language and the Tucupi implementation include some components of temporal reasoning. These features are still in a tentative stage.

The representation language allow the modeler to specify that the target activity A must be preceded by the end of B in less than say 10 hours. And that after the end of A, C must be started in no less that 20 hours but no more than 3 days. The implemented system already deals in a limited fashion with temporal limits on activities, but although the system does not perform full temporal constraint satisfaction reasoning, it is yet unclear to us how to fully characterize the temporal reasoning that is indeed performed. For example, the constraint language allow one to define temporal limits for preconditions, such as

```
rule: r5
target: C
precondition: B [10]
```

which state that B must have ended within 10 hours of the start of C. Such information is displayed to the controller, as the answer to the *what-if(C,c45,w2)* query. Also when computing the forced activities of the target C *now*, if B has ended in more than 10 hours ago, the system will determine that B must be executed again. Since B must be restarted, some of its precondition, in particular the ones with temporal limits, may no longer support the new execution of B, and they will have to restart also. The system will also compute them as forced activities for the target C.

But for example, when C *really start*, the system does not perform any check if B's precondition is still valid. The reason is that the server does not *start* the activity, it is only *informed* when the activity stared!

In standard workflows and in our constraint based workflow without the temporal information, once an activity becomes enabled, it remains enabled. The new situation, with temporal limits, seems to require the Tucupi server to give up the client-server approach and require that the starting of an activity must necessarily "go through" the system. In this case, the system can control at the time when the controller wants to start the activity if it is still enabled or not. Such change in the system is simple but it moves the system from a "workflow as just a helper" to some combination of helper and dispatcher. It is not yet clear to the developers the consequences of such decision.

Finally, also as a limitation of the currently implemented temporal reasoning component, computing forced activities is only performed if the web of preconditions (or postconditions) is a tree, that is, C may have temporal preconditions on B, and B may have temporal preconditions on A, but for example A cannot have a postcondition on some activity that is also a postcondition of B or C.

5 Related work

To the authors knowledge there are very few works on constraint or even logic based workflow representation formalisms. The closest work to this research is the one reported in [5, 4], in which a flexible workflow is defines as a set of workflow segments (in a graphic notation). A full workflow is constructed by "gluing" these workflow segments. A strong limitation of these works refers to the absence of an activation mechanism and the definition of post conditions.

The work reported in [9] also has similarities with this one. There, a temporal logic is used to represent the temporal ordering among activities in a workflow and some of the flavor of using a logic language that do not over-specify the constraint. In that work, a non-monotonic logic is used to infer which activities are enabled and thus can start immediately. In our work we use a simpler rule that all activities that have all preconditions fulfilled are enabled (modulo the time limits), and thus can start.

Paul Dourish et al. [2] describes *Freeflow*, a prototype workflow system that uses constraints as a workflow model. Similarly to Tucupi, in Freeflow the user has total control over actions in system, but does not have a controlled way of override a constraint. In Freeflow, the system only alerts the user about the existence of constraint and all the user has to do is accept or not execute the action in presence of a constraint.

Workflow with time limits between activities are also not common in the literature ([3, 6] are some of the examples). It possible that in business applications such temporal limits are less common, and thus such requirement has not been contemplated by the workflow literature. By expanding the areas of possible application of workflow technologies (for example into health care) it becomes clear that such requirements are useful.

6 Conclusion and future work

The major extension of the framework described above is to include controlled overriding of constraints. In an exceptional case, it may be necessary to violate the rules expressed in the constraints. For example, if an emergency operation is needed, some of the preconditions of this target activity may be overridden, by a controller whit the proper authorization. WRBAC [7] already includes the concept of overriding (executor) constraints for dealing with exceptional cases, and clearly such idea can be extended into the process definition itself.

Finally, the temporal reasoning component will probably include a full temporal constraint solver algorithm, in order to deal with non-tree graphs of precondition and postconditions. And very likely the server will become more of a dispatcher system in which activities are started by the system and thus the preconditions checks are performed when the activity is about to start.

Acknowledgments The authors would like to thank the comments and contributions of the anonymous reviewers.

References

1. *Role Based Access Control: Features and Motivations*, IEEE Computer Society Press, New Orleans, Louisianna, December 1995.
2. P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw. Freeflow: mediating between representation and action in workflow systems. In *Conference on Computer Supported Cooperative Work CSCW96*. ACM, November 1996.
3. J. Eder and E. Panagos. Managing time in workflow systems. In L. Fischer, editor, *Workflow Handbook 2001*, pages 109–132. Future Strategies INC, 2000.
4. P. Mangan and S. Sadiq. On building workflow models for flexible processes. In *The Thirteenth Australasian Database Conference ADC2002*, Melbourne, Australia, 2002.
5. P. J. Mangan and S. Sadiq. A constraint specification approach to building flexible workflows. *Journal of Research and Practice in Information Technology*, 2002.
6. O. Marjanovic and M. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems*, 1(2), 1999.
7. name withheld. W-RBAC: a workflow secutiry model incorporating controlled overriding of constraints. submitted, 2003.
8. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
9. J. Wainer. Logic representation of processes in work activity coordination. In *Proceedings of the ACM Symposium on Applied Computing, Coordination Track*, volume 1, pages 203–209. ACM, ACM Press, March 2000.