Ad-hoc Workflow: Problems and Solutions

M. Voorhoeve and W. van der Aalst

Dept. of Mathematics and Computing Science
Eindhoven University of Technology
Eindhoven, The Netherlands, 5600MB

Abstract

The paper introduces ad-hoc workflow, *adding flexibility to traditional workflow. A problem that stems from the added flexibility is the need to support end-users in the selection and modification of the process for a specific case. We propose a class of Petri nets to describe workflow processes, featuring* safeness *and guaranteed termination. A set of transformation rules with sufficient power for this class is given that can be implemented in a graphical editor. A second problem is monitoring the work being done. The solution here is to approximate the states of the cases being treated by the states of a few standard cases.*

1 Introduction

Workflow management systems support the daily operation of business processes by taking care of the logistic control of work (cf. WFMC[7], Koulopoulos[4], Ellis/Nutt[2]). Current workflow products support production workflow, where cases are handled according to a fixed definition of the tasks to be performed and their order. Production workflow is characterized by a high frequency and a high level of standardization. The flow of cases can be monitored closely and the occurrence of bottlenecks and slack can be identified and acted upon.

Groupware systems support less structured cooperative work. Here the tasks within a case and their order are not fixed, but can be added and modified as the case proceeds within the organization. The added flexibility has its price, though, as it becomes harder to support and control the ongoing work.

In this paper, we introduce the term *ad-hoc workflow* for processes between the extremes sketched above. Each case is derived from a *template* process that can be modified to meet specific needs. The templates do not prescribe in detail how cases are to be handled, but allow a certain degree of flexibility.

The organization and distribution of work in *processes* belongs to the realm of concurrency theory. We use Petri nets in this paper to describe concurrent processes. Petri Nets (cf. Reisig[5]) offer a model for concurrency that is simple and easily explained to non-specialists. We very briefly describe the Petri net concepts that are of importance to this paper.

A net consists of *nodes* that are connected through directed *arcs*. There are two kinds of nodes: *transitions* (depicted as rectangles) and *places* (depicted as circles); arcs only connect nodes of different kinds. A state or *marking* of a net is a bag (multiset) of *tokens* (depicted as dots). Each token belongs to a place in the net. Markings are related to one another by the *successor* relation; given a marking S, a transition t may *fire*, leading to a new marking S'. This new marking is obtained by removing a token along each incoming arc of t (if this is impossible t may not fire) and adding a token along each outgoing arc. The reflexive-transitive closure of the successor relation is the *reachability* relation. A marking with no successors is said to *deadlock*.

Nets can be structured by hierarchical decomposition into *subnets*. Petri net models for processes can be constructed directly or indirectly from other formalisms (e.g. process calculi like CCS).

2 Nature

Ad-hoc workflow is based on *process templates*. These templates provide the procedural backbone that can be filled in and varied upon to accommodate the requirements of individual cases. Hierarchy is of prime importance. The higher level templates typically allow for little variation, whereas the lower levels tend to be case dependent and can be modified as the case proceeds.

An example is given to illustrate the concept. The "WWWizz" agency offers support to companies for presenting themselves on the Internet. The agency gives courses, develops company-specific style guidelines and develops and maintains web sites. Figure 1 gives the top-level template for its activity.

Figure 1 contains two subnets. Prospective customers enter the acquisition subnet (*acq*). They then either leave through the *file* transition or become customers upon entering the *cons* subnet. After leaving this subnet, the clients have a web site and guidelines that can be maintained

0-8186-8147-0/97 $10.00 © 1997 IEEE

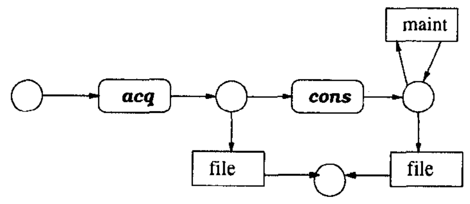

Figure 1: WWWizz top-level template

(*maint*) for some time, until the case is closed.

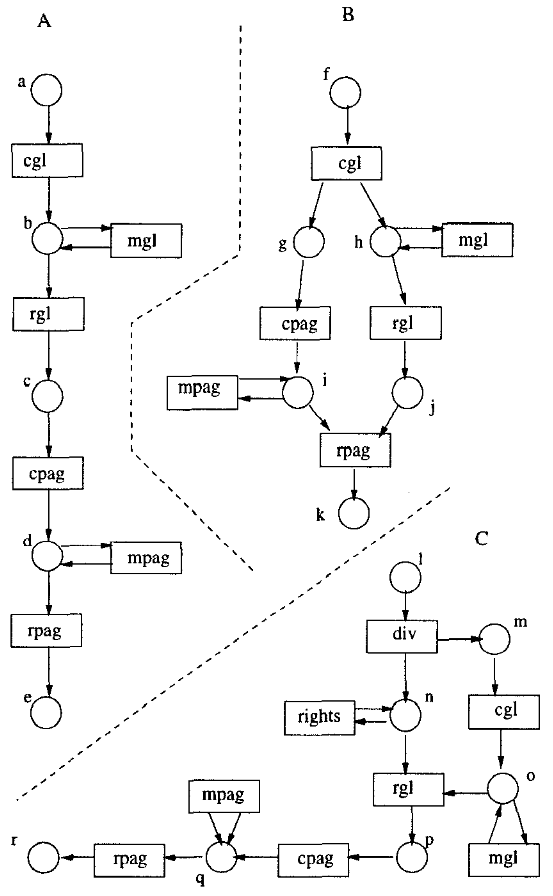

Figure 2: WWWizz (*cons*) template and modifications

Case-specific variations appear on the lower levels. Figure 2 gives the construction (cons) subnet template (A) with modifications (B, C). The template prescribes a guidelines creation phase, followed by guidelines maintenance and release, after which web pages are constructed, maintained and released.

The modification B allows page construction to start after the initial guidelines creation to get a site on the web as soon as possible. C has a negotiation phase for the inclusion of material belonging to third parties before guidelines release.

The template in ad-hoc workflow is a kind of *reference model* for the process to be executed. The template can be adapted for specific cases at any moment during the case's processing. This may involve changing the order of the tasks to be executed (like net B in Figure 2), adding (like net C in Figure 2) and removing tasks. Templates may even contain "generic tasks" that may be instantiated with whatever process, allowing a free exchange of work (i.e. groupware) at some stages.

Clearly, ad-hoc workflow requires frequent definition and modification of processes, which is an error-prone activity. So good process definition support is necessary. Another problem is the way to trace, track and manage the cases flowing through the organization. Fixed process specifications help in assessing the state of the cases in order to identify bottlenecks and take measures to resolve them. In ad-hoc workflow this becomes harder, due to the large variation of processes.

3 Process definition support

As indicated above, process definition support should enable end users to modify template processes in order to fit the needs of a specific case. A graphical and easy-to-understand definition formalism for processes (like Petri nets) is important here. As described in Aalst[1], workflow processes can be represented by a class of Petri nets called WF nets.

When modeling a process by a WF net, its transitions represent the tasks to be performed, whereas its places represent conditions that enable them. The reachable markings represent the possible states of the process. A WF net W must possess places i and o as its only source resp. sink node. Its initial marking I consists of a single token in i. Likewise, the terminal marking O consists of a single token in o. No marking may be reached that contains a token in o but O. Clearly, O *deadlocks*; this is interpreted as *successful termination*. Any other deadlock marking is interpreted as an error. From any marking S reachable from I, the terminal marking must be reachable, so erroneous deadlocks may not occur. In addition, for every transition t in W there exists a marking S reachable from I such that t can fire.

The *safe* WF (SWF) nets have the additional requirement that every marking reachable from I contains at most one token per place. Algorithms that check the WF or SWF properties of nets do exist; in fact a tool (WOFLAN[3]) has been built around them. Note that the example nets in Figures 1 and 2 are SWF nets.

We suggest the following strategy for defining SWF nets serving as process templates and adaptations. Point of departure is a library of basic SWF net templates. From them, new SWF nets can be derived by modifying and combining them in prescribed ways.

A first construction that comes to mind is *refinement* (see

Figure 3), substituting in an SWF net V a transition t with one input and one output place by another SWF net W. The entry place of W is fused with the input place of t and its exit place with the output place of t. The inverse operation replaces an SWF subnet by a transition.

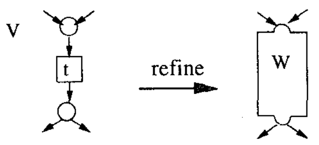

Figure 3: Refinement

Refinement has the property that applying it in either direction upon SWF nets results in an SWF net. The same does not hold for WF nets, as shown by the following example (Figure 4). Here a non-safe WF net V is shown containing a transition t. Refining t with the SWF net W results in a non-WF net containing an erroneous deadlock.

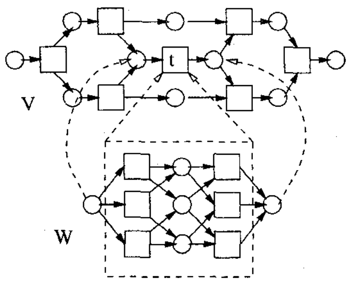

Figure 4: Non-WF net resulting from refinement

The *reduction* rule complements refinement (see Figure 5). If two nodes a, b of a net W have the property that b is the only output of a and a the only input of b, then the nodes a and b may be removed from W, adding arcs from the input nodes $e_1, \ldots, e_n$ of a to the output nodes $x_1, \ldots, x_k$ of b. The inverse of reduction is called *extension*.

The reduction and extension rules allow refinement for transitions with any number of input and output places, like in Figure 6. First, the extend rule is applied, adding a transition with a single input and ouput place. Next, the refine rule is applied and finally the reduce rule is applied. The net result is a generalized refinement.

A second group of rules is depicted in Figure 7. The *andsplit* rule splits a place into two places, duplicating the incoming and outgoing arcs. The *orsplit* rule does the same with transitions. Finally the *iterate* rule adds a transition connected to one and the same place.

Like before, the rules can be applied both ways. With refinement and reduction, all kinds of derivations can be made. In Figure 8 such a derivation is depicted, replacing a

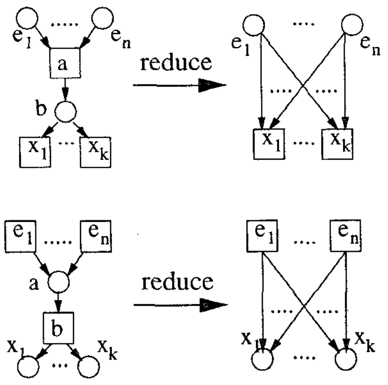

Figure 5: Reduction

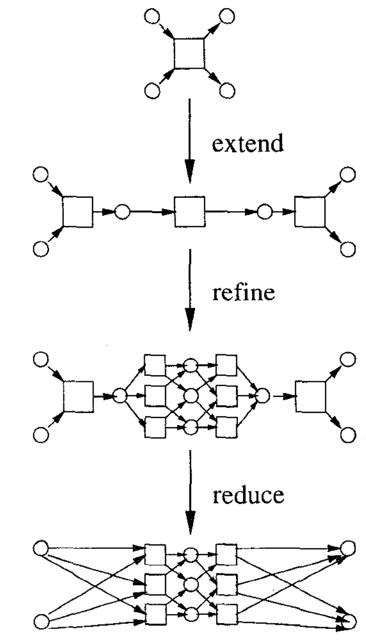

Figure 6: Refinement of arbitrary transition

task by two tasks in parallel, with a common start and ending.

Each rule correponds to a process operator from calculi like CCS and CSP. Refinement corresponds to substitution of a process for an action. Extension corresponds to sequential composition, orsplit to choice, andsplit to free merge and iteration is a special case of recursion.

To complement the above constructions, tasks may be synchronized. We distinguish two forms of synchronization, depicted in Figure 9. One-way synchronization *ssyn* prescribes tasks a, b to be performed in a fixed order. The andsplit rule in Figure 7 can be considered a special case of one-way synchronization, as it involves adding a place. Dual synchronization *dsyn* prescribes tasks a, b to be performed simultaneously, thus becoming a composed task c.

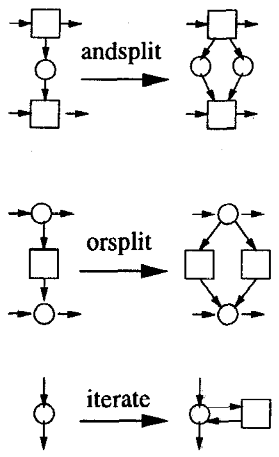

Figure 7: Split and iterate rules

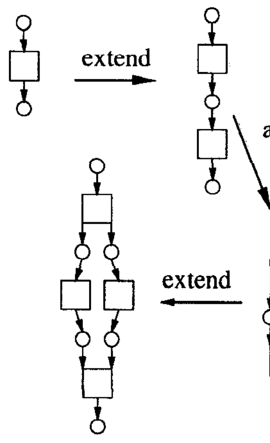

Figure 8: Example derivation

Synchronizing an SWF net may result in a non-SWF net, so checks are necessary here.

A process specification session starts with selecting a template process. From the template, refinement is possible by pointing at a transition and selecting an appropriate building block. Conversely, an SWF subnet may be indicated and shrunk into a single transition. Likewise, the other rules can be invoked, indicating the subnet and building blocks that they have to operate on.

Nets thus created can be saved to use as future building blocks. The analysis tool is used to ensure preservation of the SWF property if necessary. Organization-specific rules may be added e.g. to disallow the removal of certain vital tasks from a template process.

4 Control

Controlling the flow of work is based on reports about the progress of cases. A detailed report may be a viable approach in production workflow, but will become too large

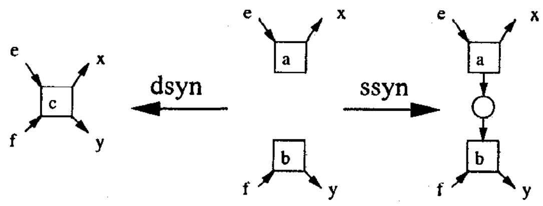

Figure 9: Synchronizing tasks

in ad-hoc workflow due to the large variation in processes.

In order to make reports understandable, the process template is used as a reference model for reports. For each case, its progress *with respect to* the template is monitored. The construction rules from the previous section except inverse synchronization allow the derivation of a function F between the states of the modified net M and its template N with the following properties. Let I_M, I_N, O_M, O_N be the initial and terminal states of M and N. Then $F(I_M) = I_N$ and $F(O_M) = O_N$. Furthermore, if a state S' is a successor (one step) of S in M, then $F(S')$ is reachable (zero or more steps) from $F(S)$ in N.

Note that it is possible to modify a template in several ways and arrive at the same end result. Different roads may result in different correspondence functions F (having the same domain and range). The correspondence may be more or less accurate, i.e. the number of steps (firings of transitions) to get from s to s' may differ more or less with the number of steps to get from $F(s)$ to $F(s')$ and the nature of these steps may differ too. The closer the modified net stays to the template the more accurate the correspondence becomes.

Given a template N, a modification M and a correspondence function F, we can approximate a state of M in N by means of F. By superposing (c.f. Voorhoeve/Aalst[6]) the approximations of the states of the cases derived from N, a manager gets an impression of the work in progress. Of course, other reports are rewuired as well. The important feature is that the manager only needs to know the template processes.

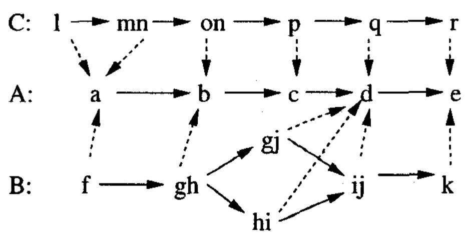

Figure 10: Relations between states in Figure 2

As an example, let us take the situation depicted in Figure 2. The states of the three nets, the reachability graphs (sclid arrows) and the functions F (dashed arrows) are depicted in Figure 10. In Figure 11 it is shown how the states of three processes (A, B and C) are superposed in a single report featuring the template process. This report fairly describes the work done so far on the cases and the work yet to be done.

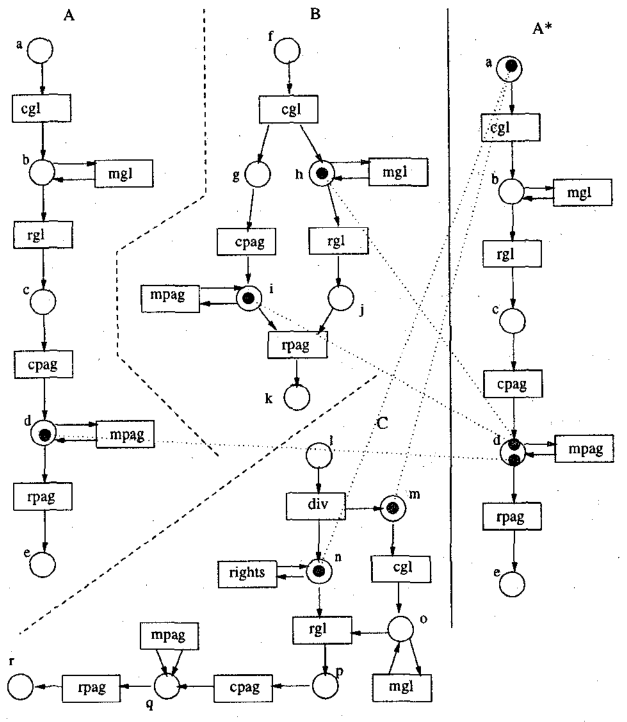

Figure 11: Individual cases states and superposition

5 Conclusion

Ad-hoc workflow is a challenging subject. Many shortcomings of current workflow management systems can be ascribed to their lack of flexibility, resulting in models with too many alternatives or ill-defined tasks. By incorporating ideas like the ones in this paper, a generation of flexible workflow management systems can be created that allow organizations to face the ever-growing demands of present-day society.

References

[1] W. van der Aalst. Verification of Workflow Nets. In *Application and Theory of Petri Nets 1997, 18th. International Conference, Proceedings*, Lecture Notes in Computer Science (to appear), Toulouse, France, 1997. Springer–Verlag, Berlin, Germany.

[2] C.A. Ellis and G.J. Nutt. Modelling and enactment of workflow systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993, 14th. International Conference, Proceedings*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer–Verlag, Berlin, Germany, 1993.

[3] D. Hauschildt, E. Verbeek, and W. van der Aalst. Woflan: A Petri-net-based Workflow Analyzer. Computing Science Reports (to appear), Eindhoven University of Technology, 1997.

[4] T.M. Koulopoulos. *The Workflow Imperative*. Van Nostrand Reinhold, New York, USA, 1995.

[5] W. Reisig. *Petri Nets*. Springer–Verlag, Berlin, Germany, 1985.

[6] M. Voorhoeve and W. van der Aalst. Conservative Adaptation of Workflow. Computing Science Reports 96/24, Eindhoven University of Technology, 1996.

[7] WFMC. Workflow Management Coalition Terminology and Glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels, 1996.