# An Agent Enhanced Framework to Support Pre-dispatching of Tasks in Workflow Management Systems

Jianxun Liu[1], Shensheng Zhang[1], Jian Cao[1], and Jinmin Hu[2]

[1] CIT Lab, Dept. of Comp. Sci. and Eng., Shanghai Jiao Tong University,
Huasan Road 1954, Shanghai 200030, P.R.China
{Ljx70@263.net}
[2] Department of Computer Science, University of Twente, Netherlands

**Abstract.** In a traditional workflow management system, it's usually assumed that: 1) a task can start to execute if and only if all the pre-conditions of that task are satisfied (such as all information it requires are prepared well); 2)a data object of a task can be released if and only if that task is completed. In practice, these assumptions are strict to some extent. It's shown from the example in this paper that even partial pre-conditions of a task is satisfied, that task can be started to work. Thus overlapping execution of tasks can be achieved. Aiming at this goal, this paper introduces first the concept of task pre-dispatching into WfMS and then presents a multi-agent enhanced WfMS framework to support it. A formalized workflow model, which can support the idea, and some implementation considerations are analyzed, too. With this approach, the duration of a process can be shortened. Even in the worst case, i.e., the overlapping of execution cannot be achieved, the pre-dispatching mechanism can still act as a messenger to inform actors when some tasks will arrive.

## 1 Introduction

A workflow is a collection of cooperating, coordinated tasks(activities) designed to carry out a well-defined complex process, such as trip planning, health care and business processes[1]. WfMS(Workflow management system) is a system that completely defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic[2]. When a business process is represented in a corresponding workflow, the number of tasks and their relationships in the workflow are dependent on a workflow designer's policy and workflow system environment[3].

Workflow has also a strong temporal aspect: activity sequencing, deadlines, routing conditions, and scheduling all involve the element of time[4]. Pozewaunig et al[5,13] mentioned time management issues in workflow management systems such as computing activity deadlines, checking time constraints, and monitoring escalations by extending PERT. While activity deadlines in [5] were statically set at workflow build-time, Panagos et al [6] studied a way to manage dynamic deadline adjustment of activities and escalations. Son et al [3] addressed a suitable scheme that can maximize the number of workflow instances satisfying the given deadline. They first presented a method to find out a set of critical activities, and then developed a method to determine the minimum number of servers for the critical activity such that this activity should be finished without delay for a given input arrival rate. Hai et al[7] proposed a timed workflow process model through incorporating the time constraints, the duration of activities, the duration of flow (where flow denotes the passing of control or data from activity to activity.), and the activity distribution with respect to the multiple time axes into the conventional workflow processes. This extension enables their proposed approach to a planning tool for shortening the execution duration of global workflow applications. These approaches aimed at optimizing the static workflow process (through distributing works in different time zone to reduce workflow instance duration) as in[7], reducing the escalation cost[6], or increasing the processing capacities for certain activities[3]. However, they neglected one situation, i.e., the processing capacities for an activity may not be fully utilized all the time, even that for the critical activities. Thus, a question if there is still space to reduce the workflow instance duration and to improve the processing efficiency of WfMS should be asked. To this question, an idea of task pre-dispatching is presented in this paper. When a task is processed, the actor for its successors will be informed to prepare if it is free. Hence, some overlapping execution of tasks can be achieved. The whole lifecycle of the process can be shortened and the efficiency will be improved. Although some similar ideas are studied and used in shop-floor process control[9], computer architecture[10] and network application(where the key idea of pre-fetching is to take advantage of the idle periods to fetch the files that will very likely be requested in the near future, so that the users average waiting time can be reduced[11].), few researchers applied it to workflow management systems either in literature or in workflow products.

The rest of the paper goes as follows. Through the analysis of an example process, section 2 introduces the concept of task pre-dispatching in WfMS. A multi-agent enhanced WfMS architecture and a formalized workflow model, which supports this concept, and some implementation considerations are presented and analyzed in section 3. Finally, section 4 concludes this paper and mentions out some further studies.

## 2 The Concept of Task Pre-dispatching in Workflow

To introduce the concept of task pre-dispatching in workflow, let’s first take a simple clinical management process as an example. This process is composed of four tasks (S and E are dummy tasks which denote start and end of the process separately. Arrowed edge denotes the control flow between tasks and dotted line from “Diagnosis” to “Cashier” and “”Getting Medicine” denotes the data flow between these tasks, i.e. both “Cashier” and “Getting Medicine” require the information, prescription, generated by “Diagnosis”). After a patient comes to the hospital and finishes “Registration”, a “Diagnosis” task is activated, in which a doctor will prescribe for this patient at his computer. The patient then pays for his prescription at cashier room, and only after he has paid, he can get the medicines from medicine warehouse.

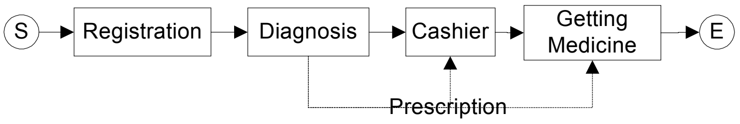

**Fig. 1.** A simple clinical process for patients in a hospital

In a workflow, a task has the following properties: task name, task type(subflow, atomic flow, etc.), pre- and post-task conditions and other scheduling constraints. And when a task can start to execute is dependent on the pre- and post task conditions[2]. While In traditional WfMS, the following assumptions are held:

- A task can be started if and only if all the pre-conditions of that task are satisfied (such as all information it requires are prepared well, the related events happened.);
- A data object used or generated by a task can be released if and only if that task is completed or aborted.

In practice, these assumptions are strict to some extent. For example, as in Fig.1, the task “Getting Medicine” can be started to collect medicines when the information, prescription, is released, even while “Cashier” is not finished. However, if the “Cashier” is not finished, the patient is not allowed to get his medicine, i.e. “Getting Medicine” cannot be finished at all until all its pre-conditions are satisfied. Thus, we define pre-dispatching of tasks in WfMS as that starting to execute a task instance when only partial pre-conditions on it are satisfied. Here, we should make an assumption that the actor of a task instance holds the knowledge what can do and what cannot do according to the satisfied pre-conditions. Otherwise, it may lead to some errors due to the unsatisfied pre-conditions. Nevertheless, with a multi-agent enhanced WfMS framework, the assumption can be easily achieved.

## 3 A Multi-agent Enhanced WfMS Framework

According to the assumption in the previous section, the actor of a task instance should hold the knowledge what can do and what cannot do according to the preconditions satisfied. This means the actor should have some extra intelligence to determine how the task can be divided into several steps and which steps in the task can be executed when the task is pre-dispatched. To support it, we present a multi-agent framework to enhance WfMS. For each actor, a SA(software agent) is associated. The actor executes the tasks of the workflow process; the agent assists and presents the actor in the system. Fig.2 shows the architecture.

**Definition 1.** A SA is a three-tuple, SA=(C, E, KB), where C denotes the communication module, which enables the SA to communicate with other SAs, the WfMS and end users; E denotes the environment module which contains information of each agent’s operational characteristics such as their business objectives; KB represents knowledge module which will be detailed next.

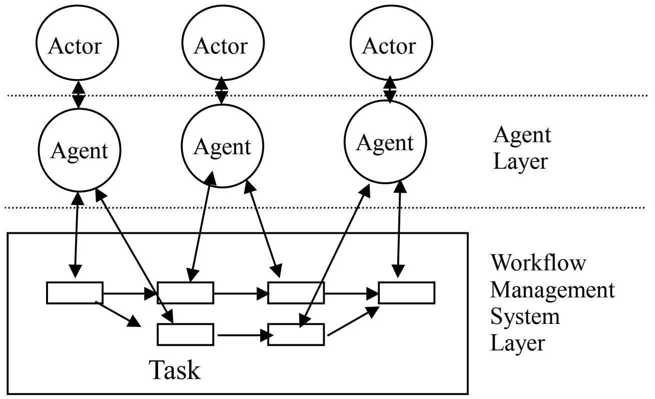

**Fig. 2.** An agent enhanced WfMS architecture

The KB of a SA includes: 1)some basic data and knowledge belong to the actor it represents, such as the personal schedule of the actor, the workload of the actor; 2) the data and knowledge that a SA needs to carry out a task, such as the state of a task instance, the constraints imposing on the task and some rule-based knowledges; 3)the knowledge that a SA needs to implement the coordination and cooperation with other SAs or WfMS.

In this architecture, we assume that only after a task has been pre-dispatched and assigned to a SA, the SA can then exchange information and events with WfMS about this task. Because the SA has the knowledge of carrying out the task, thus the SA knows which steps of a task can be executed and which can't when the task is pre-dispatched and assigned to the SA to execute. With this approach, the actor can keep from making mistakes resulted from the incompletely satisfied pre-conditions of a pre-dispatched task. For instance, as in Fig.1, when the information, prescription, is ok for a patient and the "Getting Medicine" is pre-dispatched, the SA which represents the medicine warehouse man determines whether it should display this information on the screen according to the actor's policy and the workload of the actor. When this task shows up on the screen, the actor can go to collect different medicines. However, the patient can't take the medicines away while his payment is not finished, because the SA keeps the knowledge that when "Getting Medicine" task is in pre-dispatched state, a patient can't take his medicines away.

### 3.1 A Workflow Model Supporting Pre-dispatching of Tasks

In order to support the task pre-dispatching, a workflow model, which has some extensions to the model proposed by WfMC[2], is presented as follows:

**Definition 2.** A pre-dispatching workflow model *Pre-W* is a two-tuple, *Pre-W = (TS, D)*, where *TS* is a set of tasks, $TS = \{T_1, T_2, ......, T_n\}$, $n \in N$, *N* is a natural number; *D* is the bi-relation among tasks.

**Definition 3.** The task layer numbers are assigned starting from the end, but task "E" does not have a layer number because it is a dummy task. Therefore, in Fig.1, the task in the bottom layer, "Getting Medicine", is defined to be in layer 1 and the layer number of "Cashier" is 2. In this way, each task can be assigned layer by layer. The higher layer number should be chosen if there are conflicts when deciding layer numbers[12]. For example, while a task T has more than two immediate successors(in Fig.1, "Cashier" is the immediate successor of "Diagnosis"), the layer number of T is assigned the maximal layer number among these immediate successors pluses 1.

**Definition 4.** A task T is the atomic scheduling unit of a workflow engine. T=(Id, Pre-trigger, Layer, Stride, KB, $t_{ed}$, $t_{ec}$,…), where Id is the identification of the task; Pre-trigger represents the triggering logic (pre-conditions) of the task; Layer is just the task layer number defined in definition 3; Stride represents how far from its' running predecessors by layer this task should do pre-dispatching computation. For example, Stride=2 means that when the task with layer number equal to T.Layer + 2 is running, T should be done pre-dispatching computation; KB denotes that the knowledge or guide lines for the actor as how to do this task; $t_{ed}$ denotes the estimated duration of the task; $t_{ec}$ represents the estimated time point when the task can be activated, i.e. the time point when all its pre-conditions are satisfied. The other attributes of a task such as the roles, are all left out consideration in this paper.

**Table 1.** Pre-triggers for tasks in Fig. 1

| | |
|---|---|
| Registration | ON Event S |
| DO Action ST("Registration") | |
| Diagnosis | ON Event END("Registration") |
| DO Action ST("Diagnosis") | |
| Cashier | ON Event END("Diagnosis") |
| DO Action ST("Cashier") | |
| Getting Medicine | ON Event END("Cashier")<br>DO Action ST("Getting Medicine")<br>ON Event Released("Diagnosis", "Prescription")<br>DO Action PreD ("Getting Medicine") |

* Event S means the event of starting the process, END(T) denotes the event of the completion of task T, Released(T,x) denotes task T has released information x. Action ST(T) means start to execute T and PreD(T) means pre-dispatching T.

Though a task is an atomic scheduling unit of the workflow, from the actor's point of view, it's still possible to be divided into many steps. For example, the Task "Getting Medicine" can be divided into two steps: "Collecting Medicine" and "Giving to the patient". Here may turn out a question, why do not further divide such a task into several tiny tasks while modeling, for instance, into two tasks to achieve such concurrency? The reason here is that 1)the integrity of a task may be destroyed and the process will become more complex, which makes the management more difficult; 2) It's not easy to model the physical and social world around us well, because the modeling of business process is dependent on a workflow designer's policy, knowledge level, and workflow system environment[3].

**Definition 5.** Pre-trigger is a set of ECA rules which determine the state transition and the action of a task instance such as when the task should be started or be pre-dispatched.

**Definition 6.** Each ECA rule r is a triple, *r=(e，C，A)*, where *e* is the event; *C* is the set of conditions, which represent the different situations in the system environment; *A* is a set of actions or operations. The formal representation of an ECA rule is as follows:

```
Rule  (rule_name)
   ON Event (Event_name)
   WITH (condition_exepression)
   DO (ActionSet)
```

For instance, the pre-trigger for tasks in Fig.1 is shown in Table 1. In the table, only task "Getting Medicine" has set up a trigger to pre-dispatch itself. We assume that event End(T) imply that the information generated by T has been released.

To support task pre-dispatching, a new state, Pre-dispatched, is added to the state set of a task instance defined by WfMC[2]. They are as follows:

- Waiting - the task within the process instance has been created but has not yet been activated (because task entry conditions have not been met) and has no workitem for processing;
- Pre-dispatched - A workitem has been allocated. The workers can get information about the task and prepare for it. However, the task entrance condition is not satisfied and the task cannot be finished at all unless it turns into Ready state. For example, in the clinic management process, a patient can't take his medicine if he has not paid money, though these medicines have been collected together from cabinets by warehouseman.
- Ready - the task entrance condition is satisfied, and a workitem is allocated.
- Running - a workitem has been created and the task instance is just for processing.
- Completed - execution of the task instance has completed (and the entrance condition of its successors will be evaluated)

The transition of the states is described in Fig.3. When a task is instantiated, it goes to Waiting state (no workitem allocated). At this time, if the task is pre-dispatched, its state would be changed into the Pre-dispatched state (workitem has been allocated, and some steps can be processed). When all the pre-conditions of the task are satisfied, the state is changed from Pre-dispatched or Waiting state to Ready state and the task instance will wait for actor to do it. As long as the actor begins to do that job, it goes to Running state. After the completion of the task, it goes to Completed state. Each time a state changes, an event related to this change will occur.

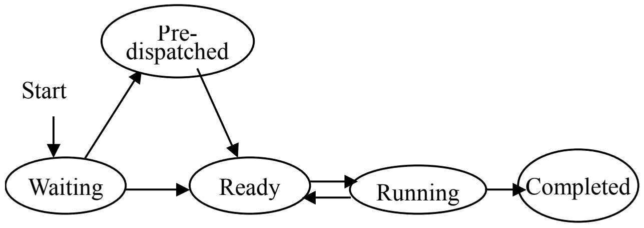

**Fig. 3.** State transitions for task instances with pre-dispatching

**Definition 7.** The state set of a task, S={"Waiting", "Ready", "Pre_dispatched", "Running", "Completed"}.

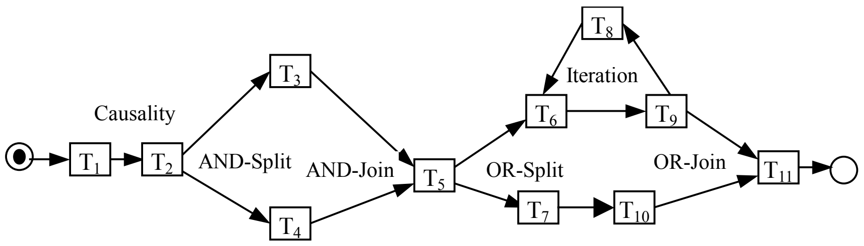

**Fig. 4.** An example workflow process

**Definition 8.** The dependency constraint, *D*, among tasks is a bi-relation on *TS*, $D : TS \times TS$.

$\forall (d_{ij} :< T_i, T_j >) \in D,\ d_{ij}$ is the probability of task $T_j$ being the immediate successor of task $T_i$. $d_{ij}$=1 represents that $T_j$ is the immediate successor of $T_i$, i.e., after the completion of $T_i$ , $T_j$ will be executed; $d_{ij}$ =0 represents that $T_j$ is not the immediate

$$D = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \\ T_{10} \\ T_{11} \end{array} \left( \begin{array}{ccccccccccc} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 & T_8 & T_9 & T_{10} & T_{11} \\ & 1 & & & & & & & & & \\ & & 1 & 1 & & & & & & & \\ & & & & 1 & & & & & & \\ & & & & 1 & & & & & & \\ & & & & & 0.5 & 0.5 & & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & 1 & \\ & & & & & 1 & & & & & \\ & & & & & & & 0.5 & & & 0.5 \\ & & & & & & & & & & 1 \\ & & & & & & & & & & \end{array} \right)$$

**Fig. 5.** Dependency constraint matrix

```
// Scheduling algorithm for task pre-dispatching
Begin
  STS = GetDownstreamTaskSet($T_{cur}$, $T_{cur}$.Layer);
    //To obtain downstream task set of $T_{cur}$
  for each task T∈STS
    if Tcur.Layer - T.Layer  <= T.Stride then
      Check(T); // function Check() will determine
            //if the pre-conditions for pre-dispatching
            //in T.Pre-trigger are satisfied,
          //if it's satisfied, T will be pre-
dispatched.
End.
Function GetDownstreamTaskSet(x, l)
// x denotes task number,
// l denotes the layer number of task x
Begin
  set dss to empty;
  //dss denotes the set of direct successor tasks of x
  // In the following, d[x,j]∈D,
  //D denotes the matrix of dependent relation
  // between tasks, n denotes the number
  //of tasks in a workflow process.
  if l>0 then
  Begin
    for j= 1 to n
      if d[x,j]=1 then // d[x,j] denotes $d_{xj}$
        dss = dss ∪ GetDownstreamTaskSet (j,l-1);
  End
  else
    dss=dss ∪x  //add task x to the task set
  Return dss;
End;
```

**Fig. 6.** Scheduling algorithm for task pre-dispatching

successor of $T_i$ ; $d_{ij} = p$ denotes that after the completion of $T_i$ , the possibility of executing $T_j$ is *p*. Under this situation, there exist many possible choices(branches) after $T_i$ . And as process continues, $d_{ij}$ will be changed according to the conditions and environments. When $d_{ij}$ becomes 1 finally, it is sure that $T_j$ will be executed after the completion of $T_i$. Following this method, it's very convenient to predict the direction of branches in a process. For the workflow process in Fig.4, the dependency constraint *D* is presented as a matrix shown in Fig. 5 (in this matrix, blank means zero, and the possibility, *p,* in each OR-branches assumes to be 0.5). It's easy to obtain the immediate successors of $T_i$ by examining all the elements in the row to which $T_i$ belongs in the Matrix.

## 3.2 Implementation Considerations

In the workflow engine, we can use the scheduling algorithm shown in Fig.6 to pre-dispatch tasks according to the proposed model. This scheduling algorithm is triggered whenever a task is started. The algorithm first obtains the downstream task(successors) set of the current task $T_{cur}$. Then for each task, *T*, in its downstream task set, it first determines if the distance, $T_{cur}$.Layer-T.Layer, between $T_{cur}$ and T is within T.Stride. When it is true, it then determines if the pre-dispatching conditions in T.Pre-trigger is satisfied. If it's satisfied, *T* will be pre-dispatched and the estimated time point, T.$t_{ec}$, is set up. Please refer to [13] for how to estimate the time point. GetDownstreamTaskSet(*x*), where parameter *x* represents task number, is a recursive function to search the successors of a task according to dependency matrix *D*. Only when $d_{ij}=1$ (denotes $T_j$ is sure to be executed immediately after $T_i$), can $T_j$ be added to the task set.

The action, PreD (*T*), will update states as follows: if *T* is in Waiting state, then just set its state to Pre-dispatched state. If *T* is not instantiated, it should be instantiated at first, turned into Waiting state, and then set to Pre-dispatching state.

## 4 Conclusion and Further Study

So far workflow has become a leading tool in modeling enterprise business rules by taking advantage of continuous advancement of information technology[8]. Workflow also has a strong temporal aspect. Recently some researchers have paid attention to it, such as reducing the workflow instance duration, improving the efficiency of WfMS and time constraints. This paper introduces the concept of task pre-dispatching into WfMS in order to improve the efficiency of WfMS. The idea proposed here is based on the fact: it is possible to start a task even when partial pre-conditions of it are satisfied. Thus, some overlapping execution of tasks in a workflow process instance can be achieved. The whole lifecycle of the process can be shortened and the efficiency will be improved, too. A formalized workflow model which supports the idea is presented. With a multi-agent enhanced WfMS architecture, it is possible to make the pre-dispatching mechanism run smoothly without leading to errors, because the SA keeps the knowledge what can do and what can't do when a task is pre-dispatched. Some extra benefits can be achieved through this agent enhanced architecture, such as cooperation between actors. With the pre-dispatching mechanism, even in the worst case, i.e., the overlapping of execution cannot be achieved, it can still act as a messenger to inform the actor when a task will arrive.

However, there are still many research questions remaining ahead, such as optimization and exception handling. For example, as in Fig.1, when the medicine warehouse man has collected the medicine but the patient changes his/her idea and goes off the hospital without "Cashier", an exception is occurred, which may result in some losses and the rollback of task is needed. These questions should be further studied.

## Acknowledgements

This work was supported by China National Science Foundation under grant No: 60073035, 69974031, and by the China Super Science and Technology Plan 863/CIMS under the grant No: 2001AA415310, 2001AA412010.

## References

1. Lee, M.K., Han, D.S. and Shim, J.Y. Set-based access conflict analysis of concurrent workflow definition. Information Processing Letters. 80(2001):189-194
2. WfMC. Workflow Reference Model. http://www.wfmc.org/standards/docs. Jan 1995.
3. Son, J.H. and Kim, M.H. Improving the Performance of Time-Constrained Workflow Processing. The Journal of Systems and Software. 58(2001):211-219
4. Chinn, S.J. and Madey, G.R. Temporal Representation and Reasoning for Workflow in Engineering Design Change Review. IEEE Transactions on Engineering Management. Vol.47(4). 2000. pp:485-492
5. Pozewaunig, H., Eder, J. and Liebhart, W.. ePERT: extending PERT for workflow management systems. In: The 1$^{st}$ European Symposium in ADBIS. vol.1,1997:217-224
6. Panagos, E. and Rabinovich, M. Reducing escalation-related costs in WFMs. In: NATO Advanced Study Institute on Workflow Management Systems and Interoperability. 1997:106-128
7. Hai, Z.G., Cheung, T.Y., and Pung H.K. A timed workflow process model. The Journal of Systems and Software. 55(2001):231-243
8. Avigdor, G. And Danilo, M. Inter-Enterprise workflow management systems. 10th International Workshop on Database & Expert Systems Applications, Florence, Italy, 1-3 September, 1999 : 623-627
9. Applequist, G., Samikoglu, O., Pekny, J. and G.Reklaitis. Issues in the use, design and evolution of process scheduling and planning systems. ISA Transactions. Vol.36(2), 1997, pp. 81-121
10. Steven, P. and David, J. Data Prefetch Mechanisms. ACM Computing Surveys. Vol.32, No.2, June 2000. pp174-199
11. Jiang, Z.M. and Kleinrock, L. An Adaptive Network Prefetch Scheme. IEEE Journal on Selected Areas in Communications. Vol.16(3), 1998, pp 358-368
12. Yan, J.H and Wu, C. Scheduling Approach for Concurrent Product Development Processes. Computer in Industry. 46 (2001):139-147
13. Eder, J., Panagos, E., Pezewaunig, H., and etc. Time management in workflow systems. 3d Int. Conf. on Business Infomation Systems, pp. 265-280, Invited paper. April 1999.