Agent-oriented Model for Managing Long-lived Transaction, Based on Work-flow and Task-graph

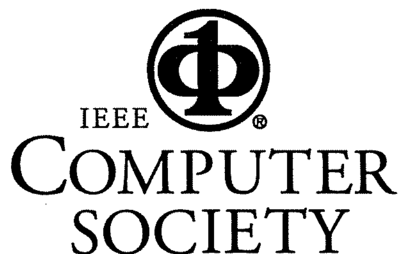
Toyohide Watanabe

Reprint

Proceedings of the

Third International Conference on Computational Intelligence and Multimedia Applications
ICCIMA '99

New Delhi, India
23 - 26 September, 1999



Washington ◆ Los Alamitos ◆ Brussels ◆ Tokyo

PUBLICATIONS OFFICE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1314 USA

© Copyright The Institute of Electrical and Electronics Engineers, Inc. Reprinted by permission of the copyright owner

Agent-oriented Model for Managing Long-lived Transaction, Based on Work-flow and Task-graph

Toyohide Watanabe
Department of Information Engineering,
Graduate School of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

Abstract

The management of long-lived transaction is one of interesting topics as one application of databases for long-term enterprise requests. Some methods have been proposed to manage this type of transactions well: nested transaction model, SAGAS model, etc. However, the models are not always successful to control various classes of long-lived transactions effectively. In this paper, we describe first our 3-layer model of long-lived transaction which was developed to avoid such troublesome problems, and address the management strategy to be constructed under 3-layer model, based on work-flow and task-graph.

1 Introduction

The subject about management of long-lived transaction is one of interesting topics as one application of databases for long-term job which is composed of many interrelated requests[1]. The basic issue is to control and execute mutually related requests effectively in accordance with their execution orders and relationships. Until today, some methods have been proposed to cope with this issue: the nested transaction model[2] and SAGAS transaction model[3]. The nested transaction model organizes some related transactions hierarchically, arranges the execution dependency among individual transactions systematically and makes the execution control easily. In this model, there is a drawback on which the successor depends completely on the predecessor and is disturbed if the predecessor could not finish its own work well. While, SAGAS transaction model introduces the compensate transaction in order to recover the drawback in the nested transaction model. In this model, the idea of compensate transaction is appreciable, but in practice it is not easy to specify the compensate transactions for individual transactions.

In this paper, we address an agent-oriented transaction management means to overcome successfully drawbacks in the previous models. In particular, we discuss the framework for managing long-lived transaction under 3-layer transaction model[4], and describe the management method, based on the work-flow and task-graph[5].

2 Framework

We organize the management structure for long-lived transaction as 3-layer model hierarchically. Figure 1 shows our 3-layer model conceptually. The top layer manages the long-lived transaction as a user request wholly. This layer is called the long-lived transaction layer. Namely, this layer controls the execution of jobs requested from users uniformly. The middle layer manages the composite units, which are executable in sequencial control. This layer is called the transaction layer. Namely, this layer manages the work derived mandatorily from the long-lived transaction layer and controls the execution units, which are atomic in concurrent control. Finally, the bottom layer executes the atomic execution units in practice. This layer is called the action layer.

0-7695-0300-4/99 $10.00 © 1999 IEEE

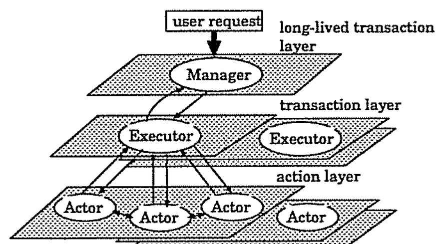


Figure 1: **3-layer transaction model**

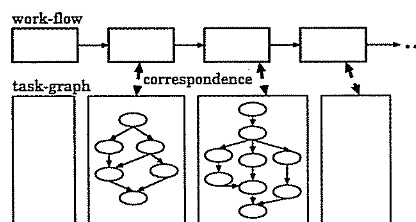


Figure 2: **Work-flow and task-graph**

We assign agents with particular roles to individual layers with a view to making the execution control among layers successful. Figure 1 illustrates these agents in each layer: Manager, Executor and Actor. Manager in the long-lived transaction layer is a responsible entity for a given long-lived transaction and controls Executors sequencially in centralized organization. Executor in the transaction layer is a responsible entity for a given transaction (or work) inquired from Manager and controls the execution among Actors concurrently. Actors located in the action layer execute individual tasks in practice.

In this framework, the dependency problem among transactions is solved by the functionality of autonomous agent. Also, the compensation problem for failure transaction is supported complementarily by the control mechanism between Manager and Executor because Manager enables/disables appropriate Executor and Executor enforces/disforces executing Actors through the centralized execution control, respectively.

The long-lived transaction is composed as a sequence of interrelated composite transactions, and also the transaction is composed as an ordered set of atomic actions on the basis of hierarchical management architecture. Thus, the long-lived transaction is defined as a linear list of independent transactions, and the composite transactions are sequencially executed. Namely, Executors are exclusively existing in system. While, the transaction is defined as a multi-way graph of dependent/independent actions, and atomic Actors are executed concurrently. Namely, Actors are concurrently existing in system.

A linear list of independent transactions is represented as the work-flow: the work specifies the requisite conditions, procedural behaviour, goal and so on for the corresponding transaction. Also, the multi-way graph of dependent/independent actions is represented as the task-graph: the task specifies the requisite conditions, procedural behaviour, goal and so on for the corresponding action. Namely, the work-flow and task-graph are collections of formally organized processes for long-lived transaction and transaction, respectively. Figure 2 shows the relationship between work-flow and task-graphs. Each task-graph is organized by 1-1 relationship for the corresponding work.

Our work-flow is a global specification of long-lived transaction, while our task-graphs are local specifications of individual transactions. Manager handles the work-flow directly and controls Executors one by one. Of course, Manager controls backword or forward the execution points of the work-flow when some failures were encountered. Executor deals with the task-graph appropriately, associated with the work, and controls individual Actors in accordance with the task-graph, dynamically.

3 Work-flow and Task-graph

The work is defined as one independently executable process and is commonly organized as a message for the corresponding Executor. Manager is responsible for the transaction management, and composes the executable work-flow on the basis of appropriate templete of work-flow initially. After the work-flow has been organized initially, Manager controls individual Executors sequencially with respect to the work-flow. The work-flow represents

declaractively a collection of compound processes for long-lived transaction.

Task-graph is defined as a set of command messages to be applied to Actors. Although each Executor manages its own preassigned task-graph, the tasks are instanciated by receiving the requisite parameters from work.After Executor initialized the corresponding task-graph, it makes the execution of tasks effectively in accordance with the task-graph. The task-graph represents various relationships among individual tasks so as to make the execution efficiency successful. The typical relationships among tasks are as follows:

1) *Conditional start*: This relationship between tasks A (upper) and B (lower) means that B is started when the condition attended to the edge is satisfied. In this case, the condition can be specified by globally reserved variables "n-return" and "b-return" or special task variable "finish". "finish" keeps the execution result-value "commit" or "abort" of the previous task. Figure 3 shows such relationships. In case that condition is not specified, the relationship is interpreted as normal dependency.

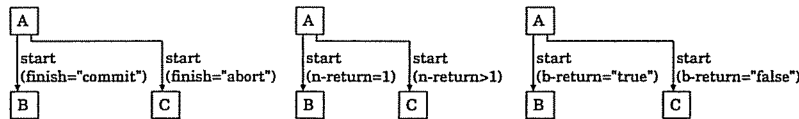


Figure 3: **Conditional start between tasks**

2) *Exclusive execution*: This relationship among tasks B, C and D means that B, C and D are concurrently executed, but C and D are exclusive(in Figure 4(a)). Of course, when tasks B, C and D must be exclusive, this is shown in Figure 4(b).

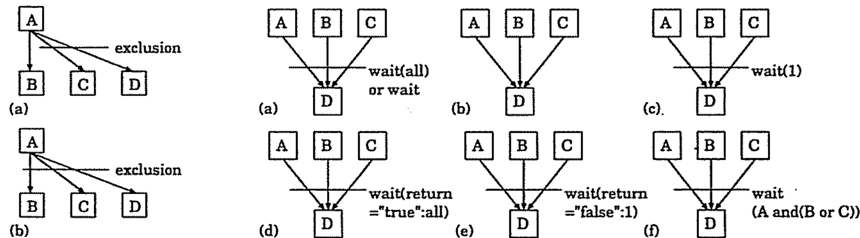


Figure 4: **Exclusive execution among tasks**

Figure 5: **Synchronization**

3) *Synchronization*: This relationship among tasks A, B, C and D means that A, B and C enforce the execution of D in accordance with condition. The condition is the number of input tasks, or the value for a globally reserved variable "return":
   (1) wait(all) or wait: for completion of all predecessors ((a) or (b) in Figure 5);
   (2) wait(1) : for completion of only one task from all ((c) in Figure 5);
   (3) wait(return="true": all): for completion of all tasks whose return values are "true"((d) in Figure 5);
   (4) wait(return="false": 1): for completion of one task whose return value is "false" ((e) in Figure 5);
   (5) wait(A and (B or C)): for completion of task A and completion of task B or C ((f) in Figure 5).
4) *Conditional stop*: This relationship between tasks A and B means that B should be stopped when the condition was satisfied. The condition description is the same as that in the conditional start. Figure 6 shows such a situation.

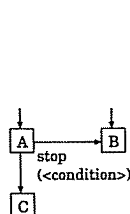


Figure 6: **Conditional stop**

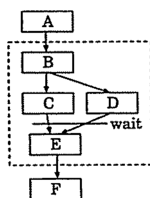


Figure 7: **Block structure**

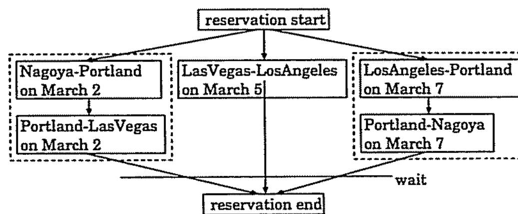


Figure 8: **Task-graph of fleight reservation**

5) *Block structure*: The block represents a subtask-graph for a complex task. In this case, this complex task is manipulated as a control unit for "commit" or "abort" though this unit is adaptable to only one task. Figure 7 shows such a structure.

4 Example

Here, we consider an example: management in travel agency. We assume simply that a long-lived transaction for travel management is divided into independent transactions such as planning, reservation for transporations, reservation for hotels, reservation for optional events, confirmation, payment and issue of tickets. Manager deals with this work-flow in order to manage the request of a user smoothly. Of course, the backward control is applied if necessary. When the result in one transaction was cancelled by changing user's plan, the flow control goes back to the transaction related to the most initially generated result.

Next, we consider one transaction as a work: reservation for transporations. In this case, reservation for flights from Nagoya to Portland on March 2, 1999, Portland to Las Vegas on March 2, Las Vegas to Los Angeles on March 5, Los Angeles to Portland on March 7, and Portland to Nagoya on March 7. Figure 8 illustrates the task-graph for this work. Of course, this task-graph manages Executor "reservation for transporations", instanciated from the class "reservation for flights", and also individual tasks are assigned to the corresponding Actors: in this case, the same Actors are created.

5 Conclusion

In this paper, we proposed our 3-layer model for managing a long-lived transaction on the basis of agent-oriented concept. In particular, three different agents in individual layers take different roles to manage the hierarchical transaction structure effectively and to control the execution successfully without depending on the hierarchical structure. Additionally, we introduced the work-flow and task-graph. These control mechanisms make the execution efficiency sufficiently and also make the execution control flexibly.

References

[1] A.K.Elmagarmid: "*Database Transaction Models for Advanced Applications*", Morgan Kaufmann Pub. (1990).

[2] J.E.B.Moss: "*Nested Transactions: An Approach to Reliable Distributed Computing*", Cambridge Pub. (1981).

[3] H.G.Molina and K.Salem: "SAGAS", *Proc.of ACM SIGMOD'87*, pp. 249–259 (1987).

[4] M. Asano, T. Ushiama and T. Watanabe: "Managing Long-lived Transaction on Agent-oriented Model", *Proc. of VSMM'96*, pp. 485–489 (1996).

[5] A. Cichocki, A.S. Helal, M. Rusinkiewicz and D. Woelk: "*Workflow and Process Automation: Concepts and Technology*", Kluwer Academic Pub. (1998).