Towards Context-Aware Ubiquitous Transaction Processing: a Model and Algorithm

Feilong Tang[†,‡], Song Guo[‡], Minyi Guo[†], Minglu Li[†] and Cho-li Wang[§]
[†]Department of Computer Science and Engineering
Shanghai Jiao Tong University, Shanghai 200240, China
[‡]School of Computer Science, The University of Aizu, Japan
[§]Department of Computer Science, The University of Hong Kong, China

***Abstract*—Transaction management for mobile and ubiquitous computing aims at providing mobile users with reliable services in a transparent way anytime anywhere. To make such a vision a reality, transaction processing for the mobile and ubiquitous computing needs to adapt to the runtime environments dynamically. However, most existing mobile transaction models do not consider the context-based transaction management. In this paper, we propose a context-aware transaction model and context-driven coordination algorithms. They are built on an event-context-action mechanism, enabling the transaction processing to adapt well to dynamically changing transaction context. The simulation results have also demonstrated that our model and algorithms can significantly improve the successful commit ratio under unstable context conditions.**

I. Introduction

Mobile and ubiquitous computing (MUC) is a new distributed computing paradigm. Through MUC, people can get online access to their preferred services even while moving around, by sharing computing, communication and information services anytime anywhere. Open MUC environment is prone to failures caused by devices, applications, networks, and basic services of ubiquitous systems. However, most present researches and projects on MUC only focus on communication mechanisms (e.g., new communication protocols tailored for ubiquitous environments) and ubiquitous devices (e.g., with more powerful functions, smaller size and less cost)[2]. This motivates us to investigate the reliability support for advanced ubiquitous applications.

Transaction processing has been widely used to transparently guarantee system consistency in various distributed environments. To hide the complexity of service processes from users as much as possible, ubiquitous systems have to dynamically adapt to changing contexts and intelligently handle failures and recovery [6]. Therefore, the advanced transaction management is another key technology[2] to ensure the reliability of MUC in a transparent way. The necessity to set up reliable MUC systems has been discussed in literature, e.g., in [10]. Furthermore, as pointed out in [2,9,10], the traditional mobile transaction proposals are not directly applicable to MUC transactions because MUC allows users to move freely while enjoying preferred services anytime anywhere[10]. Under such circumstances, nodes, data and services that can be directly accessed keep changing with the movement of users.

Transaction management is highly related to computing paradigms. High mobility of MUC environments presents severe challenges to MUC transaction management in the following aspects.

Context-aware transaction model. Transaction context includes information from physical space, information space and human activities. Context awareness is at the center to enable transparent ubiquitous transaction services. Therefore, the MUC transaction model should have the abilities to adapt to dynamical transaction contexts, and thus it can provides general support for various ubiquitous applications.

Context-driven dynamic transaction management. The mobility of MUC users makes transaction context, e.g., network connection and bandwidth, continuously changing. In order to improve the successful ratio and efficiency of transactions and to reduce users' intervention as much as possible, transaction management should be aware of context changes, and intelligently optimize the distribution and execution modes of ubiquitous transactions.

In this paper, we investigate how to solve the above challenges. Firstly, we propose a context-aware MUC transaction model that handles transactions in an event-context-action manner. Then, we present the context-driven coordination algorithms that achieve fast and efficient adaption for transaction processing. Our objective is to provide a context-aware and transparent transaction service for advanced ubiquitous applications. Distinguishing from previous work, our model and algorithms focus on adaptively adjusting transaction execution patterns according to current context, which significantly improves successful commit ratio under unstable environments.

The remainder of this paper is organized as follows. Section II gives a review of related work. Section III proposes a context-aware MUC transaction model. In Section IV, we investigate context-driven transaction coordination algorithms. Experiments and evaluation on our model and algorithms are reported in Section V. Finally, Section VI concludes this paper with a discussion on our future work.

II. Related Work

Existing mobile transaction models mainly focus on the issues like network disconnection, concurrency control, replication, hand-off and restricted resources[4], paying little or no

attention to the context-awareness which has become a crucial part of modern mobile applications.

Intermittent connection is the most serious challenges to mobile transaction processing. To recover from node disconnections, CLCP [1] uses multiple coordinators to achieve robust and failure-tolerant atomic commit. In [5], a transaction processing framework is proposed to support frequent disconnections of mobile databases. The main idea is that transaction execution can be done at both the BS (base station) and MHs(mobile hosts). Similarly, HiCoMo(High Commit Mobile)[3] keeps base tables for base transactions and aggregate tables for HiCoMo transactions. Concurrency control is another important issue for mobile transaction management. Park et al.[7] proposed an optimistic concurrency control method based on a random back-off technique. [8] proposes a multiversion transaction processing approach and a deadlock-free concurrency control mechanism based on a multi-version two-phase locking scheme integrated with a timestamp approach.

Transaction management in ubiquitous environments needs to detect transaction context, such as connectivity and network bandwidth, and then flexibly decides how to coordinate subtransactions. It has not achieved by existing proposals. This paper is motivated to solve this key issue, especially on the adaptive transaction model and coordination algorithms.

III. Context-Aware Transaction Model

A. Context of MUC Transactions

The significant difference between MUC and traditional mobile computing is that MUC frees users from the complexity of service processes as much as possible by automatically collecting context and adaptively adjusting execution policies. So context awareness is a prerequisite to the MUC transaction processing.

Transaction context includes information from physical space, information space, and human activities related to transaction processing. More specifically, MUC transactions mainly concerns on the following context.

- *Wireless network(WN)*, with attributes connectivity, bandwidth, delay, losing ratio, cost and stability
- *Mobile device(WD)*, with attributes computing_capacity, available_memory, available_battery, available_data, available_cache and security
- *Location*, with attributes longitude and latitude
- *User*, with attributes profile, purpose and requirements
- *Time*, with attributes start_time and ending_time.

B. Context-Aware Transaction Model

A context-aware transaction model for ubiquitous environments can be formulated by the following definition.

Definition 1 (MUCT)**.** *An MUC transaction (MUCT) is a 6-tuple* $MUCT = (T, CT, ECA, D, TS, FS)$*, where:*

T=$\{T_i \mid 1 \leqslant i \leqslant n\}$ and CT=$\{CT_i \mid 1 \leqslant i \leqslant n\}$ are the set of subtransactions in an MUCT and the set of compensating transactions for the subtransactions, respectively, in which n is the number of the subtransactions in an MUCT. Each subtransaction T_i ($T_i \in T$) can associate at least one compensating transaction CT_i such that CT_i can semantically undo the affects of T_i submitted previously.

TABLE I
Transaction state description.

Symbol	*State*	*Description*
I	initiation	transaction does not start to execute
E	executing	transaction is executing and has not committed
S	submitted	transaction has successfully committed
F	failed	transaction has failed to commit and been rollbacked to the previous state
C	compensating	compensating transaction is being executed

ECA=$<ECA_i \mid 1 \leqslant i \leqslant n>$ is the list of ECA (event-context-action) rule descriptors, where ECA_i is the rule descriptor of the subtransaction T_i. ECA_i has higher priority than ECA_{i+1}.

Each ECA_i=$<E_i, C_i, A_i>$ is also a list of 3-tuple: event, context and corresponding action, which describe multiple context-based execution policies for the subtransaction T_i, and are described as follows.

- E_i=$\{E_{ij}\}$: a set of events that occur during the execution of T_i.
- C_i=$\{C_{ik}\}$: a set of context associated with a subtransaction T_i. We also take C_i as the conditions that must be met for the execution of T_i. C_i covers five dimensions: WN, MD, Location, User, and Time as described in the last subsection. Note that C_i changes dynamically.
- A_i=$\{A_{il}\}$: a set of corresponding actions. For example, a user wants to reserve flight tickets from traveling agent A, and the communication link is disconnected at that time. Once such an event is detected, transaction manager may try to connect another traveling agent B to resume the ticket reservation.

D is a set of dependencies between T_i and T_j (T_i, $T_j \in T$). We define two kinds of dependencies: *successful submission dependency* ($\lessdot_s$) and *failed submission dependency* ($\lessdot_f$). The dependency $T_i \lessdot_s T_j$ means that T_j cannot be executed until T_i successfully commits, while $T_i \lessdot_f T_j$ denotes that T_j can start its execution only if T_i fails to commit.

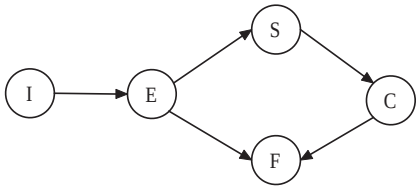

Fig. 1. State conversion diagram of MUC transactions.

$TS = \{S_1, S_2, \ldots, S_n\}$ is a set of states of all subtransactions in a MUC transaction. S_i, the state of subtransaction T_i, is in one of five possible states: I, E, F, S and C (see Table I). Each subtransaction starts with state 'I' and ends with state 'S' (if it commits successfully) or 'F' (if it fails to commit). Fig.1 illustrates the state conversion.

FS is a set of acceptable final states. An MUC transaction may have multiple proper final states that an user is ready to

accept, probably with different priorities.

C. A Case Study

In this section, we describe how to model MUC transactions in a specific scenario. We consider a patient John, driving a car equipped with a mobile device(MD), needs to find a hospital in emergency. MD first queries the nearest hospital to find whether it satisfies medical conditions (T_1) for John. If the hospital qualifies, MD sends John' current physical status to the hospital to prepare corresponding medical services(T_2), and then queries the public traffic information service to discover the best path to that hospital based on current road states(T_3). If T_1 fails, MD subscribes the medical service from the Health Service Central Hospital(T_4), which can provide versatile medical services while probably being far away.

We model these activities as a MUC transaction such that T=$\{T_1, T_2, T_3, T_4\}$ and CT=$\{\phi, CT_2, \phi, \phi\}$, where CT_2 is the compensating transaction for canceling the reserved medical services. The dependency in T is illustrated in Fig.2.

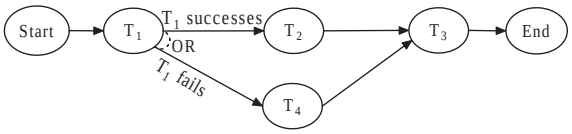

Fig. 2. A case of MPC Transactions.

ECA=<<E_1,C_1,A_1>,<E_2,C_2,A_2>,<E_3,C_3,A_3>,<E_4,C_4, A_4>> describes how to adapt to the context, where <E_1,C_1,A_1>=<<E_1,C_{11},A_{11}>,<E_1,C_{12},A_{12}>>. Specifically, <E_1,C_{11},A_{11}> means that MD successfully queries the nearest hospital if the context C_{11} is qualified, such as connectivity=connected and bandwidth = high (or medium), for the query, where E_1 means that John initiates a query to the hospital; A_{11} is the action that MD queries the hospital successfully. Accordingly, if C_{11} is unqualified, A_{12} will be executed, which means MD will contact with the Health Service Central Hospital. <E_2,C_2,A_2>, <E_3,C_3,A_3> and <E_4,C_4,A_4> are similar.

D=$\{T_1 \lessdot_s T_2,\ T_2 \lessdot_s T_3,\ T_1 \lessdot_f T_4,\ T_4 \lessdot_s T_3\}$. This dependency D specifies that this transaction may be executed in one of two sequences σ_1=$\{T_1,T_2,T_3\}$ and σ_2=$\{T_1,T_4,T_3\}$.

FS={(S, S, S, -), (F, -, S, S)}, where 'S' and 'F' stand for the successful and failed execution of the corresponding subtransaction respectively, while '-' means that the execution state of the subtransaction does not affect the decision.

IV. Context-Driven Coordination Algorithms for MUC Transactions

A global MUC transaction T=$\{T_1,T_2,\ldots,T_n\}$ is initiated by a mobile device (called *requestor*) while subtransactions are distributed to n mobile devices (called *executor*). Accordingly, our context-driven coordination approach based on the proposed MUC transaction model consists of two parts: *algorithm1* and *algorithm2*, which are respectively executed by a Coordinator in a transaction requestor and n Participants located in executors.

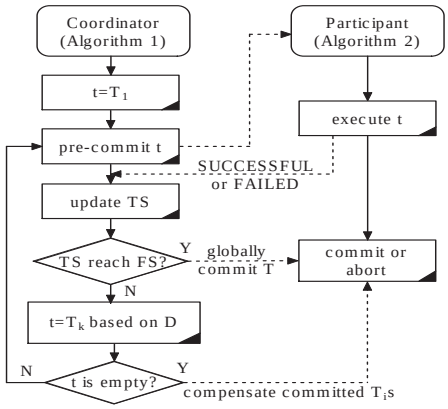

Fig. 3. Coordination flow for MUC transactions.

Algorithm 1 Coordination algorithm in a requestor

Input: $T=\{T_1, T_2, \ldots, T_n\}$, $CT=\{CT_1, CT_2, \ldots, CT_n\}$
ECA rules, dependency set D
Output: T execution result

```
1: query host H_i for executing T_i in registration center;
2: TS={I,I,...,I};
3: t←T_1;
4: while (TS⊈FS) do
5:    if (an event t.E_k occurs) {
6:       check corresponding context t.C_k;
7:       for i=0 to RetryNum do
            if (t.C_k is qualified) {
            send t to H_k;
            S_k='E';
            wait incoming message;
            if (the message is SUCCESSFUL)
               S_k='S';
            else
                if (the message is FAILED)
                   S_k='F'; }
         end for }
8:    t←(D) next subtransaction T_k;
9:    if (t = φ) break;
10: end while
11: if (TS⊆FS)
       globally commit T by the message CONFIRM;
12: else
       compensate subtransactions committed previously
       by the message CANCEL;
```

The global MUC transaction T is executed until its state TS achieves one of acceptable final states or no subtransaction exists. The order of executing subtransactions is determined by transaction dependency D. In the case studied in last section, for example, T_4 will be immediately executed if T_1 fails. The state of each subtransaction is set according to its

execution result. Finally, if one of acceptable final states has achieved, the algorithm confirms all submitted subtransactions. Otherwise, it requires subtransactions submitted previously to execute their corresponding compensating transactions. We illustrate the transaction coordination flow in Fig.3.

Algorithm 2 Coordination algorithm in executors

Input: Subtransaction T_k
Output: Execution result of T_k

```
failed:=false;
while (T_k does not finish and (not failed)) do
    execute application operations in T_k;
    if (fail to execute the operations)
        failed:=true;
end while
if (failed)
    send a FAILED message to the requestor;
else {
    send a SUCCESSFUL message to the requestor;
    wait incoming message;
    if (receive a CONFIRM message)
        report execution results to the requestor;
    else if (receive a CANCEL message) {
        execute T_i's compensating transaction;
        report T_i is cancelled;     } }
```

The algorithm1 coordinates global MUC transactions, where a subtransaction T_i is submitted to an execution node only when (1) it has not been executed(i.e., S_i='I'), (2) the corresponding event occurs, and (3) its context C_i is qualified. By the *qualified context*, we mean that the current context of a transaction T_i satisfies requirements for executing T_i.

The algorithm2 actually executes individual subtransactions, under the control of the Coordinator by interacting with coordination messages. If a subtransaction is successfully executed, it will be confirmed or compensated upon receiving the message CONFIRM or CANCEL from the Coordinator, respectively. Otherwise, it automatically rollbacks to the previous system state.

V. Experiments and Evaluation

We have implemented a simulation system to evaluate the feasibility of the proposed MUC transaction model. We have also investigated how much the successful commit ratio of MUC transactions is improved by using our context-driven coordination algorithms.

A. Experiment Environment

In our system, there are 100 self-organized mobile nodes. Each node acts as a requestor as well as an executor. More specifically, any node can not only issue global MUC transaction requests but also execute sub-transactions submitted by other nodes.

The features of node mobility are simulated by changing link states. Let wireless links between nodes disconnect in a probability *DisconnectProb*. In addition, we model the system load in the number of concurrent MUC transactions (marked as *NumMobiTran*). The MUC transactions were randomly initiated and concurrently executed in the system. Each of them consists of two subtransactions as T=$\{T_1,T_2\}$. Our current experiments concentrate on how to adapt to the dynamic network connectivity and bandwidth.

We evaluate two kinds of transactions: *context-aware transaction (WithCA)* and *non-context-aware transaction (WithoutCA)*. For WithCA transactions, if the context of a subtransaction T_i is unqualified, the Coordinator in a requestor resends T_i to other nodes for at most RetryNum (RetryNum>1) times. On the other hand, a WithoutCA transaction is dispatched only once so it would fail if at least one link among a requestor and executors is unqualified. In the following experiments, we set RetryNum=3.

B. Results and Evaluation

Because high mobility and frequent network disconnection significantly decrease the transaction commit probability, the *successful ratio (SR)* is normally used as the major performance metrics of mobil transaction models. Therefore, we evaluate MUC transactions through *SR* that is defined as a ratio of transactions submitted successfully to total transactions within a given period.

For each kind of MUC transactions, we conducted comprehensive simulations in the following various network settings.

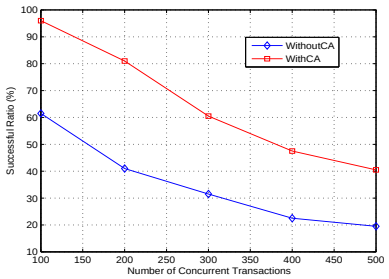

Fig. 4. SR against the number of concurrent transactions.

1) The Number of Transactions: In this experiment, we varied the number of concurrent MUC transactions from 100 to 500, where link disconnection probability is fixed at DisconnectProb=0.1. The performance results obtained for the two kinds of transactions WithCA and WithoutCA are shown in Fig.4. The SR degrades for both strategies as the transaction load increases. The reason is that higher load on the physical resources causes heavier data conflicts. From this figure, we can see that in the whole range of the transaction load, WithCA always performs better than WithoutCA. This is because WithCA may redispatch a transaction for at most RetryNun times if previous requests failed while WithoutCA sends a transaction request only once.

2) Disconnected Probability: When a link disconnects or has no enough bandwidth, the nodes connected by the link can no longer initiate transaction requests or report execution

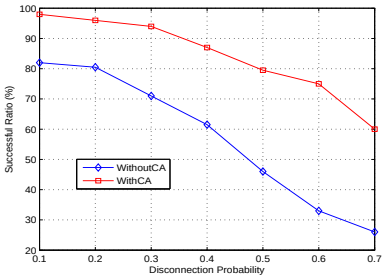

Fig. 5. SR against link disconnection probability.

results. Therefore, for both WithCA and WithoutCA transactions, link states have significant impact on the SR. In this experiment, we measure the transaction SR by varying the *DisconnectProb* from 0.1 to 0.7 with an increment of 0.1. The performance results are shown in Fig.5, where the number of concurrent MUC transactions in system was fixed as *NumMobiTran*=50.

As we can see, the performance of the system becomes worse for both execution strategies WithCA and WithoutCA as the value of link disconnection probability increases. The reason is that with the increase of failure probability of wireless links, more subtransactions can not be sent to the target nodes. However, the relative performance of WithCA to WithoutCA is not affected by such probability of wireless link failure because WithCA transactions would resend subtransactions to other nodes for more options that WithoutCA transactions do not have.

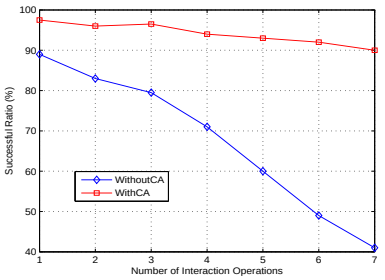

Fig. 6. SR against the number of interaction operations in a transaction.

3) The Number of Interaction Operations: Interaction operations are issued by a mobile host during its execution. As shown in the previous example, if the query for the nearest hospital is successful, the mobile device sends John's current emergency status to the hospital and queries public traffic information service to discover the best path to that hospital based on current road states. We can expect that the successful ratio of MUC transactions will become worse as such interactions increase duo to communication failures. In this experiment, we varied the number of interaction operations in MUC transactions from 1 to 7 with the increment of 1 to evaluate the performance impact of the number of interaction operations. Performance results obtained in terms of the success ratio are provided in Fig.6, where NumInteration means the number of interaction operations in a MUC transaction.

As the number of interaction operations increases, the performance of WithCA gets a little bit worse because WithCA transactions can be re-executed multiple times. By comparison, a steeper performance degradation is observed in WithoutCA strategy as the number of interaction operations increases. The reason is that WithoutCA transactions are dispatched only once so that the failure probability of these transactions is accumulated with the increasing interactions among a mobile device and ubiquitous services through unstable wireless connections.

VI. Conclusions and Future Work

We have presented a context-aware transaction model and context-driven coordination algorithms. The proposed model and algorithms can adaptively adjust transaction management policies according to the dynamic transaction context. The performance studies show that our proposal can significantly improve the transaction commit ratio of MUC transactions under unstable environments.

We are going to investigate flexible and light-weight mechanisms to automatically generate compensating transactions for MUC environments.

Acknowledgment

Feilong Tang thanks The Japan Society for the Promotion of Science (JSPS) for providing the excellent research environment for his JSPS Postdoctoral Fellow (ID No. P 09059) Program in The University of Aizu, Japan.

This work was supported by the National Natural Science Foundation of China (NSFC) with Grant Nos. 61073148 and 60773089, the National Science Fund for Distinguished Young Scholars with Grant Nos. 61028005 and 60725208, and Hong Kong RGC with Grant No. HKU 717909E.

References

[1] S.Obermeier, S.Bottcher, D.Kleine. A cross-layer atomic commit protocol implementation for transaction processing in mobile ad-hoc networks. *Distributed and Parallel Databases*, 26(2): 319-351, 2009.

[2] M.Franklin. Challenges in Ubiquitous Data Management, *Informatics*, pp. 24-33, 2001.

[3] M.Lee, S.Helal. HiCoMo: High Commit Mobile Transactions, *Kluwer Academic Publishers Distributed and Parallel Databases (DAPD)*, 11(1): 73-92, 2002.

[4] R. A.Dirckze, L.Gruenwald. A pre-serialization transaction management technique for mobile multidatabases, *Mobile Networks and Applications (MONET)*, 5(4): 311-321, 2000.

[5] Z.Abdul-Mehdi, A.B.Mamat, H.Ibrahim et al. A model for transaction management in mobile databases. *IEEE Potentials*, 29(3):32-39, 2010.

[6] R.Rouvoy, P.Serrano-Alvarado, P.Merle. Towards Context-Aware Transaction Services. In *LNCS 4025*, pp.272-288, 2006.

[7] S.Park, S.Jung. An energy-efficient mobile transaction processing method using random back-off in wireless broadcast environments. *Journal of Systems and Software*, 82(12):2012-2022, 2009.

[8] S.K.Madria, M. Baseer, V.Kumar et al. A transaction model and multiversion concurrency control for mobile database systems. *Distributed and Parallel Databases*, 22(2):165-196, 2007.

[9] N.Roy, S.K.Das. Enhancing Availability of Grid Computational Services to Ubiquitous Computing Applications *IEEE Transactions on Parallel and Distributed Systems*, 20(7):953-967, 2009.

[10] F.Perich, A.Joshi, T.Finin et al. On Data Management in Pervasive Computing Environments, *IEEE Transactions on Knowledge and Data Engineering*, 16(5): 621- 634, MAY 2004.