User-Centric Transactions for Ambient Intelligence Environments

Vasileios Fotopoulos, Apostolos Zarras, Panos Vassiliadis

User-Centric Transactions for Ambient Intelligence Environments

Vasileios Fotopoulos, Apostolos Zarras, Panos Vassiliadis

University of Ioannina, Dept. of Computer Science, Ioannina, Hellas
{vfotopou, zarras, pvassil}@cs.uoi.gr

Abstract. In this paper we investigate the concept of designing user-centric transaction protocols towards achieving dependable coordination in AmI environments. As a proof-of-concept, we propose a protocol that takes into account the schedules of roaming users that move from one AmI environment to another, to avoid abnormal terminations of transactions when the users leave an environment for short, only to return later. We compare the proposed schedule-aware protocol against a schedule-agnostic one. Our findings show that the use of user-centric information in such situations is quite beneficial.

Keywords. Ambient intelligence, transactions, two phase commit.

1. Introduction

The rapid emergence of novel technologies in the fields of mobile computing and networking fostered the transition from conventional distributed systems to mobile computing systems that consist of fixed and mobile devices (such as PDAs, Pocket PCs, smart-phones), which collaborate through wireless networking infrastructures. Going one step further, the vision of *Ambient Intelligence (AmI)* investigates the possibility of realizing mobile computing environments that are aware and responsive to the presence of people [1, 2]. AmI is based on Weiser's pioneer work on ubiquitous computing [3], which evolved later on to the concept of pervasive computing. Pervasive computing aims at a digital world, consisting of interconnected electronic devices that support the quotidian activities of people. AmI is particularly concerned by the users' experience in such a digital world. In other words, AmI puts a specific focus on the users and targets the development of *user-centric digital environments* that account for the users' needs, habits and satisfaction, while offering support that allows them to perform their everyday activities.

The vision of AmI motivates research towards coordination protocols that involve both mobile and fixed entities. In this paper, we particularly investigate the need for designing *user-centric transaction protocols* to achieve dependable coordination in AmI environments. *User-centric information can be exploited while coordinating a set of transaction participants towards avoiding abnormal transaction terminations*.

In this context, we focus on the abnormal ending of a transaction that takes place within an AmI environment, due to the fact that *one or more participating users leave the environment*. Leaving the environment means that the users' devices are no longer reachable, via the networking infrastructure that supports the transaction coordination.

The idea behind our approach is that *if there is a certain level of knowledge behind the schedule of each participating user (i.e.,, the way the user moves from one environment to another)*, *then we can exploit it to avoid abnormal transaction terminations, where a roaming user leaves the environment for short, only to return later*.

Taking a simple example, consider a conference that takes place in a number of conference rooms. Several researchers attend a technical session in conference room A (i.e., environment A). In this situation, a number of colleagues want to arrange a meeting for dinner or work after the technical session. To this end, they setup a private ad-hoc network using the Bluetooth capabilities of their mobile devices. One of them browses, using his Pocket-PC, information regarding available meeting places. His goal is to book a place at a certain time and insert a dinner meeting in the agenda applications that execute on his colleagues' laptops or Pocket-PCs. Obviously, setting up the dinner meeting involves performing a distributed transaction amongst the mobile devices that host the agenda applications. The transaction requires each participant's agenda application to execute a local transaction and verify that there are no other obligations of the participant at the meeting time. This task might take a certain amount of time to complete. Assume now that during this time period, one of the participants leaves the gathering before the transaction completes, because his talk starts at conference room B (i.e., environment B). In such a situation, typical transaction protocols would abort the transaction, wasting thus the energy resources that were spent up to this point. Nevertheless, the transaction may have a chance for successful completion if we consider that the colleagues shall reunite after the coffee break. Hence, if the transaction protocol could be enriched with such kind of user-centric information (i.e., the users schedules) and reason with respect to this information, all the work that has been performed for fixing the dinner meeting would not be wasted.

Based on the previous discussion, *the contribution of this paper consists of designing a schedule-aware protocol and comparing it against a schedule-agnostic one*. Specifically, in Section 2 we present the necessary background and state-of-the art for this paper. In Section 3 we detail the proposed protocol. In Section 4, we present our experimental results. Finally, in Section 5 we summarize our contribution and provide insights for future work.

2. Related Work & Background

The overall idea of user-centric transaction protocols and the particular protocol discussed in this paper fall in the general field of mobile transactions [4, 5]. Until now there have been various approaches for mobile transactions that can be classified with respect to the system model that they assume into 3 different categories [5]. In all of them the transaction initiator is a mobile host and the entities that comprise the data, processed during the transaction execution are fixed hosts. Moreover, in [5] the authors further identified the following more generic execution models:

1. In the first system model, transactions are initiated by mobile hosts and they aim at processing data located on other mobile hosts.

2. The second system model is the most generic one, where the execution of mobile transactions is distributed amongst several mobile and fixed hosts.

A few years ago, the previous execution models were considered as too ambitious but interesting [5]. Nowadays, these models fit perfectly to the case of AmI environments. Until now, some interesting approaches have been proposed for dealing with transactions in the context of the aforementioned execution models. In [6], for instance, the authors deal with mobile host disconnections in transactions that involve several mobile and fixed hosts by a protocol that discovers alternative mobile hosts that may replace the disconnected ones. In [7] the authors propose a protocol for transactions that span across several mobile hosts, which may move across different interconnected network cells. The main idea is to use participant-agents (i.e. proxies to participants that move to different network cells) to provide relocation transparency and timeouts to handle participant disconnections. Alternatively, in [8] the authors propose the use of a data sharing space. In this paper, we go one step further by investigating the issue of using user-centric information towards designing distributed transaction protocols for AmI environments.

The protocol that we investigate in this paper relies on the combination of two classical protocols: (a) the presume-abort 2-phase-commit protocol [9] and (b) the strict 2-phase-locking protocol [10].

In general, the execution of a transaction involves (1) an entity that initiates it (hereafter we use the term *master* to refer to the transaction initiator) and (2) entities that comprise data, processed during the transaction execution (hereafter we use the term *cohort* to refer to these entities). Typically, the transaction execution consists of an *initiation* state, during which the master invites the cohorts to participate in the transaction and the cohorts accept or deny the invitation. If all goes well, the *initiation* state is followed by an *executing* state, during which the master processes data that may be of his own, or of the participating cohorts. At the time when the master decides to complete the transaction, the presume-abort protocol takes place amongst the participants. Briefly, the presume-abort protocol comprises the two phases of the classical 2-phase-commit protocol. During the first phase, the master of the transaction sends to all cohorts a PREPARE message. Upon the reception of this message the cohorts should respond with their votes concerning the outcome of the transaction. The voting messages may be either to commit or to abort the transaction. After the voting the transaction gets into a *prepared* state and the cohorts wait for the final decision for the outcome of the transaction. The second phase of the protocol starts after the reception of all votes sent by the cohorts. If a negative vote exists, the master decides to abort the transaction, notifies accordingly all cohorts, and releases all information concerning the transaction (i.e., the transaction gets into an *aborting/aborted* state). Otherwise, if all votes were positive the master decides to commit the transaction, notifies accordingly all cohorts and waits for their acknowledgment (the transaction gets into a *committing* state). Upon the reception of the acknowledgments, the master releases all information concerning the transaction and the transaction get into a *committed* state. The presume-abort protocol further conforms to the following basic principle: *if a transaction participant tries to find out about whether a transaction was finally committed or aborted and there is no information available about this transaction, the transaction participant derives the conclusion that transaction was aborted*.

The strict 2-phase-locking protocol that we assume is a variant of the classical 2-phase-locking protocol, whose fundamental principle states that *no locks can be released until all necessary locks have been acquired from the transaction*. In the strict 2-phase-locking variant, *all locks are released at the end of the transaction*.

3. A Schedule-Aware Protocol for AmI Environments

In this section we discuss the issue of user-centric transactions in the context of AmI environments. The problem we wish to handle concerns the abnormal ending of the transaction due to the fact that a mobile user / transaction participant leaves the AmI environment where the transaction takes place. The idea behind our approach is that if there is a certain level of knowledge behind the schedule of the user, then we can exploit it to avoid the abnormal transaction termination. Based on this idea, we present a schedule-aware transaction protocol. Before presenting the protocol's internals, in Section 3.1, we start with preliminary concepts, foundations and assumptions for our problem.

3.1 Preliminaries

In this section we provide a formal definition of the entities that participate in the AmI environment, along with any assumptions made for the purpose of this paper.

We assume that an AmI environment is a set of nodes N_i in an area where they can communicate with each other. Therefore, we define the notion of AmI environment based on both an area and the available networking facilities that allow communication between a set of nodes. Our overall system model consists of a set of distinct AmI environments of nodes $\mathbf{S}=\{N_1, N_2, \ldots, N_n\}$. Communication between nodes of N_i, N_j, for all $i, j \mid i \neq j$ is not possible. We assume two kinds of nodes, (a) *fixed nodes* that are constantly part of their environment and (b) *mobile nodes*, corresponding to users that move over the set of AmI environments. At any given time point, each environment comprises its fixed nodes and a (possibly empty) set of mobile nodes that happen to be part of the environment at that moment. Each node *n* has (a) a unique node id and (b) a finite set of *records*, or *variables*, denoted as *var*(*n*), which are either read or updated in the context of a (possibly distributed) transaction. Moreover, each mobile node is characterized by a schedule that specifies its movement from one environment to another. A node's schedule is a finite list of pairs of the form (*environment, duration*) characterizing how long the node will remain in each environment. In Fig. 1, we depict he schedule of a node which is going to stay for 20 time points in environment N_1, then move to environment N_2 where it will remain for 30 time points, then return to environment N_1 for a duration of 40 time points and finally move to environment N_3 where it will stay for 44 time points.

All nodes issue flat distributed transactions, i.e., transactions composed of tasks that are executed at different nodes, with the extra assumption that each node who is requested to perform a task can execute this task locally without issuing another

(nested) transaction. Also, we assume that each transaction is executed within the context of a single environment[1].

Formally, each transaction is defined as the following tuple:

$$T = (TID, NID, MID, \{Steps\})$$

where *TID* is a unique identifier for the transaction, *NID* is the identifier of the environment within which the transaction must be executed, *MID* is the node identifier for the master node of the transaction and *Steps* is a finite list of steps (to be defined right away)[2]. Each Step is defined as a set of *actions*, with each action being a request to read or write a cohort's variable. An action is, thus, defined as the following tuple:

$$A = (CID, Action, Variable)$$

where *CID* is the node identifier of the cohort node that executes the action, *Action* belonging to the set {*READ*, *WRITE*} and *Variable* being the variable being read or written.

N_1	20
N_2	30
N_1	40
N_3	44

Figure1. Exemplary schedule of a mobile node.

3.2 The Freeze on Leave Protocol

The main thrust of our contribution lies in the exploitation of the schedules of the mobile nodes. Assume that a mobile node is about to leave an environment where it participates as a cohort to a distributed transaction. In this case, a typical transaction protocol would simply abort the transaction. Following a different direction, we build on the idea on requiring the node to notify the transaction's master on its intention to leave, instead of sending an abort message. The crux of the proposed protocol is that the master tries to find *a rendezvous, i.e., a time point and a subsequent interval where all the participants of the transaction will meet again in the same environment*. If this is feasible, then the transaction is frozen, its state is recorded at the master and it will be de-frozen again when the master's clock reaches the starting point of the rendezvous that the master has calculated. Due to this mechanism, we call this protocol *Freeze on Leave* (FOL).

Assume a transaction that takes place in environment N_1 and involves a fixed master and two mobile cohorts, m_1 and m_2. Assume that at time point τ the master receives a message from cohort m_1 that the latter is leaving environment N_1. The schedules of the two cohorts at time point τ are depicted in Fig. 2. The master, can calculate that, according to the cohorts' schedules, cohort m_1 will be back at the

[1] This particular assumption can be relaxed with straightforward enhancements in the proposed protocol.

[2] For reasons that will be apparent in the sequel, we would like to point out that it is easy to infer whether a node is mobile or fixed by its node id.

environment N_1 for the time interval [51-90] and cohort m_2 will also be back for the time interval [71-90]. The overlap of the two schedules can serve as a "rescue" interval for the successful completion of the transaction.

N_3	20
N_2	30
N_1	40
N_3	44

Schedule for m_1

N_1	30
N_3	40
N_1	20
N_3	10

Schedule for m_2

Figure 2. Schedules for mobile nodes at the time of departure of m_1.

Interestingly, the protocol does not guarantee successful completion of the transaction. The risks of failure are primarily two: (a) a cohort violates its schedule and misses the rendezvous for the frozen transaction's defreeze, or (b) the transaction cannot be completed in the common time interval of the cohorts. In both of the aforementioned cases the protocol guarantees that the transaction shall be aborted. Another issue that should be mentioned concerns the protocol's requirement that master nodes are aware of the schedules of mobile cohorts. This particular requirement raises an issue of privacy, which should be handled by the middleware that would actually realize the protocol on behalf of the transaction participants. Further details concerning this issue are out of the scope of this paper.

In the rest of this section, we organize the discussion of the internals of the Freeze On Leave protocol in two parts: first we assume that the master is fixed and following we examine the case where the master is mobile. In both cases, the reaction of the master is also dependent upon the state in which it is in.

If the master of the transaction is fixed, then it does not need to worry about its own schedule, since it will continuously be present at the environment where the transaction takes place. As already mentioned, we are particularly interested in the case where a mobile cohort sends a message *LEAVE* to the master, signifying the cohort's intention to leave the environment. Whenever the master receives such a message it checks its state. If the master is in any state before *executing*, then it assumes that no work has actually been done (and therefore worth saving) and aborts the transaction. On the other hand, if the master is in an *executing* or *prepared* state, it understands that there is a chance of salvaging the work that has been performed so far. The actions of the master depend upon its state.

A cohort leaves and the master is in executing state: In this case, when the master receives the LEAVE message from the cohort, it initiates the procedure for finding *a rendezvous*, *i.e., a common time point and a subsequent interval where all the mobile cohorts will be back in the environment again*. In case there is no such interval, the transaction is aborted as usually. If, on the other hand, such an interval exists, the master proceeds as following:

- First, the master checks whether there are steps that can be executed without the leaving cohort. If the next step requires the departing cohort, then the master node proceeds as follows:

- *it notifies all cohorts about the rendezvous* by sending to them a FREEZE message;
- if the master has received acknowledgements from the last step (i.e., read or write actions), it assumes a *hung up* state – else it assumes an *ack hung up* state until all acknowledgements arrive;

– If there are steps that can be executed without the departing cohort, then the master proceeds as follows:
  - *it notifies the departing cohort* about the rendezvous by sending to it a FREEZE message;
  - it assumes a *temp executing* state;
  - it waits for a step that requires the presence of an absent cohort to signal a *FREEZE* message *to all the cohorts* and moves to a state of *hung up* or *ack hung up*.

At the same time, when a cohort receives a FREEZE message, it moves to a *hung up* state.

The execution of the transaction continues interactively. Whenever a participating cohort returns to the environment, the master node tries to execute the next step of the transaction. If the execution of the next step is possible the master passes in a *temp executing* state and keeps up with the execution of the transaction until a step that requires a missing node; otherwise it remains in its previous state.

The overall defreeze of the transaction takes place when the rendezvous point arrives. At this point, the master checks if every cohort is present. If the rendezvous is missed, the master aborts the transaction and notifies all cohorts that are present accordingly. The cohorts that missed the rendezvous are aware of this situation; when the rendezvous is missed each one of them considers the transaction aborted.

A cohort leaves and the master is in prepared state: If the master receives a *LEAVE* message when it is in *prepared* state, it also needs to check whether it is possible to find *a rendezvous*. If such a rendezvous can not be found the transaction is aborted. Otherwise, the master (a) sends a FREEZE message *to the cohort leaving the environment* and (b) assumes a *vote hung up* state, waiting for the remaining cohorts' votes. When the master can reach a decision for the transaction, there are two cases:

- If the transaction is to be aborted, the master notifies all cohorts that are present about the decision and assumes a *partially abort* state, until the rendezvous point. At this point, the master sends an *ABORT* message to the returning cohorts and moves to an *aborted* state. Note that some cohorts may miss the rendezvous. These cohorts can not be notified by the master about the outcome of the transaction. However, since they are aware of the missed rendezvous, they shall abort the transaction by themselves.
- If the transaction is to be committed, the master moves to the *partially commit* state, until the rendezvous. At this point, the master checks if every cohort is present. If the rendezvous is missed, the master assumes an *aborted* state. As previously, the cohorts that missed the rendezvous abort the transaction by themselves. If the rendezvous is met by all cohorts the master assumes a *committing* state.

If the master of the transaction is mobile, the overall behavior of the protocol is quite similar with what has been discussed for the case where the master is fixed. Nevertheless, below we summarize the main differences that exist in the case of the mobile master:

- Whenever the master tries to calculate a rendezvous, it takes into account *its own schedule along with the schedules of the participating cohorts*.
- If the master has to leave the environment while being in the *executing* or in the *prepared* state, there is nothing particularly different from the case of a mobile cohort leaving the environment. Nevertheless, due to the fact that the master needs to organize its departure and calculate the rendezvous, the master arranges *to send a LEAVE message to itself somewhat earlier than its departure*.

4. Experiments

To assess the idea of designing user-centric transaction protocols for AmI environments we implemented a simulator and performed a number of experiments. The goal of our experimental evaluation was to compare the FOL protocol we proposed in Section 3 against a schedule-agnostic protocol. The schedule-agnostic protocol relies on the following principle: whenever the designated time interval for the staying of a mobile node at a certain environment expires, the node (a) sends a message that aborts all the transactions to which it participates, and (b) leaves the environment (possibly to join the next environment in its schedule). The main metrics for our study were the percentages of aborted and committed transactions in the case of each protocol.

Concerning our experimental setup, we assumed 3 different AmI environments, each one of which comprised 30 fixed nodes. Given these environments we performed 4 different sets of experiments where the number of mobile nodes varied as follows: 10, 15, 20 and 25 mobile nodes. The overall number of variables for the fixed nodes was 640, while the overall number of variables for the mobile nodes was 320. The variables were equally distributed among the fixed and the mobile nodes.

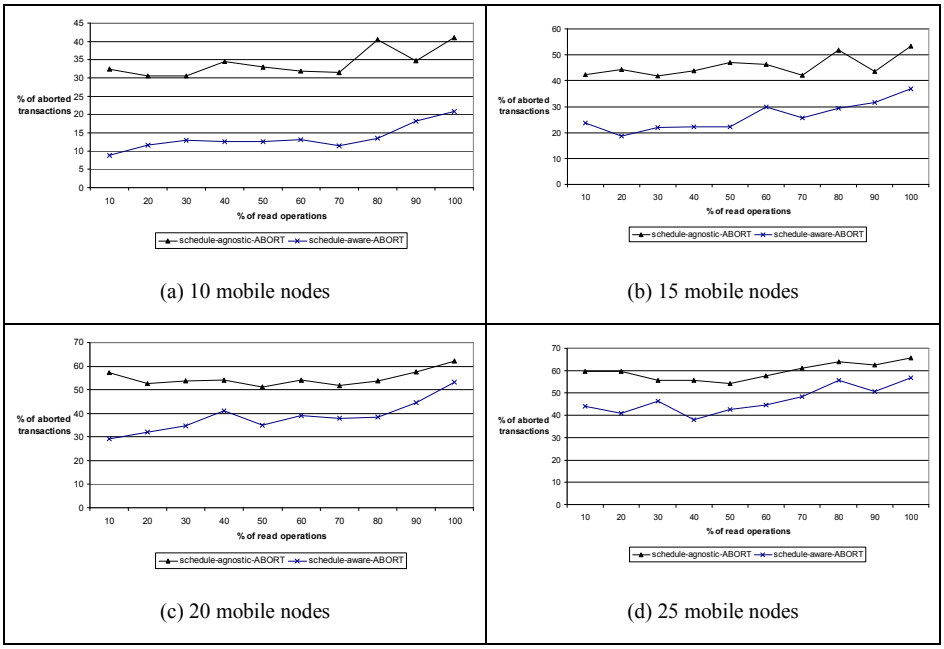
The schedule of each mobile node was randomly generated with respect to the overall simulation time which was set to 1000 time units. The average visiting time of each node in a particular environment was 50 time units (i.e., it was randomly generated in the range [40, 60]). Therefore, each mobile node performed on average 25 visits in the 3 AmI environments.

The set of transactions used in our experiments was also randomly generated. In particular the number of steps of each transaction varied in the range of [1, 20]. Hence, on average every transaction comprised 10.5 steps. The average number of actions performed on each step was 2. Each action had a probability of 0.5 to be performed on a variable that belonged to a randomly selected mobile node. In each one of the 4 different sets of experiments that we performed we varied the percentage of read actions over the total number of actions from 10% to 100%. The percentage of read operations influences the contention for locks within each node, since read operations can read-lock the same variable simultaneously, whereas write operations

lock the variables exclusively. Finally, in all our experiments, transactions were initiated in the AmI environments according to a Poisson distribution; on average, 2 transactions were initiated every 10 time units.

Fig. 3 summarizes the results we obtained. More specifically, Fig. 3 gives the percentages of aborted transactions resulted by the use of the two protocols in the 4 different configurations of our environments. In all cases, we can observe that the schedule-aware protocol exhibits a much better behavior; the percentages of aborted transactions in the case of the schedule-agnostic protocol are much higher than the percentages of aborted transactions in the case of the schedule-aware protocol. Nevertheless, as we increase the number of mobile nodes involved in the 3 AmI environments the difference between the two protocols decreases given that the probability of finding rendezvous decreases.

Concerning the percentages of committed transactions, (detailed results are not provided due to the lack of space) our main observation was that the schedule-aware protocol performs better than the schedule-agnostic one as the percentage of read operations increased. Moreover, the difference between the two protocols became clearer as we increased the number of mobile nodes involved in the environments.



(a) 10 mobile nodes

(b) 15 mobile nodes

(c) 20 mobile nodes

(d) 25 mobile nodes

Figure 3. FOL vs. a schedule-agnostic protocol: Percentage of aborted transactions.

5. Conclusions and future work

In this paper we discussed our general position that concerns the need for designing user-centric transaction protocols towards achieving dependable coordination in AmI environments. We proposed such a protocol that takes into account the schedules of roaming users that move from one AmI environment to another, to avoid abnormal terminations of transactions when the users leave an environment for short, only to return later. We compared the proposed schedule-aware protocol against a schedule-agnostic one. Our findings showed that the use of user-centric information in such situations is quite beneficial. Our results motivate further investigation of the issue of user-centric transaction protocols. Currently we focus on more stochastic approaches for defining and exploiting user centric information. Privacy is also an interesting issue involved. Moreover, our research is oriented towards the design of customizable protocols where the outcome of transactions shall be decided with respect to user-defined context rules. Finally, we envision the provisioning of middleware support for user-centric transaction protocols.

References

1. Aarts, E., Harwig, R., Schuurmans, M.: The Invisible Future: The Seamless Integration of Technology into Everyday Life. In: Ambient Intelligence. McGraw-Hill (2001) 235-250
2. Weber, W., Braun, C., Glaser, R., Gsottberger, Y., Halik, M., Jung, S., Klauk, H., Lauterbach, C., Schmid, G., Shi, X., Sturm, T., Stromberg, G., Zschieschang, U.: Ambient intelligence - key technologies in the information age. In: Proceedings of the IEEE International Electron Devices Meeting (IEDM'03). (2003) 1.1.1-1.1.8
3. Weiser, M.: The Computer of the Twenty-First Century. Scientific American 135 (1991) 94-104
4. Pitoura, E., Samaras, G.: Data Management for Mobile Computing. Kluwer Academic Publishers (1998)
5. Serrano-Alvarado, P., Roncancio, C., Adiba, M.: A Survey of Mobile Transactions. Distributed and Parallel Databases 16 (2004) 193-230
6. Younas, M., Chao, K-M., Wang, P., Huang, C-L.: QoS-aware Mobile Service Transactions in a Wireless Environment. Concurrency and Computation: Practice and Experience, 19 (2007)
7. Nouali, N., Doucet, A., Drias, H.: A Two-Phase Commit Protocol for Mobile Wireless Environment. In: Proceedings of the 16th ACM Australasian Database Conference (ADC'05). (2005) 135-143
8. Le, H.N., Nygard, M.: Mobile Transaction System for Supporting Mobile Work. In: Proceedings of the 16th IEEE International Workshop on Database and Expert Systems Applications (DEXA'05). (2005) 1090-1094
9. Mohan, C., Lindsay, B., Obermarck, R. Transaction Management in the R Distributed Database Management System. ACM Transaction on Database Systems, 11 (1986)
10. Eswaran, K.P., Gray, J.N, Lorie, R. A., Traiger, I. L. The Notations of Consistency and Predicate Locks in a Database System. Communication of the ACM, 19 (1976)