Schedule-Aware Transactions for Ambient Intelligence Environments

Vasileios Fotopoulos, University of Ioannina, Greece

Apostolos V. Zarras, University of Ioannina, Greece

Panos Vassiliadis, University of Ioannina, Greece

ABSTRACT

In this paper, the authors investigate the concept of designing user-centric transaction protocols toward achieving dependable coordination in AmI environments. As a proof-of-concept, this paper presents a protocol that takes into account the schedules of roaming users, which move from one AmI environment to another, avoiding abnormal termination of transactions when users leave an environment for a short time and return later. The authors compare the proposed schedule-aware protocol against a schedule-agnostic one. Findings show that the use of user-centric information in such situations is quite beneficial.

Keywords: Ambient Intelligence, Schedule-Agnostic, Schedule-Aware, Transactions, Two Phase Commit

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 (Aarts, Harwig, & Schuurmans, 2003; Weber et al., 2003). AmI is based on Weiser's pioneer work on ubiquitous computing (Weiser, 1991), 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.

DOI: 10.4018/jaci.2010100106

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. 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.

1. RELATED WORK AND 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 (Pitoura & Samaras, 1998; Serano-Alvarado, Roncancio, & Adiba, 2004). 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 (Serano-Alvarado et al., 2004). 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, Serano-Alvarado et al. (2004) 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.

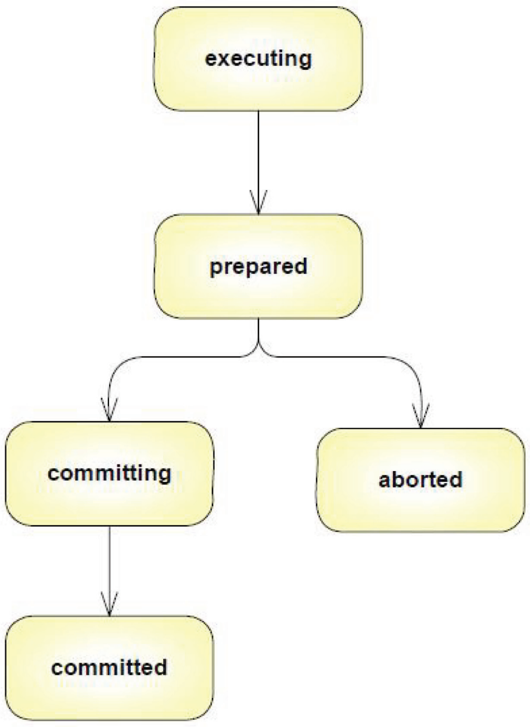
A few years ago, the previous execution models were considered as too ambitious but interesting (Serano-Alvarado et al., 2004). Nowadays, these models fit perfectly to the case of AmI environments. Until now, some interesting approaches have been proposed for dealing with transactions in the context of the aforementioned execution models. For instance, Bobineau, Pucheral, and Abdallah (2000), proposed a one-phase commit protocol for transactions involving mobile and fixed hosts, where the voting phase is eliminated and the master announces its own decision to the transaction participants; the decision is taken based on the master's perception on the successful execution of the individual transaction steps. Kumar, Prabhu, Dunham, and Seydim (2002), proposed TCOT where the master employs timeouts towards deciding about the outcome of a particular transaction. Younas, Chao, Wang, and Huang (2007) dealt 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. Nouali, Doucet, and Drias (2005) proposed 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, Le and Nygard (2005) proposed the use of a data sharing space. Finally, Böttcher, Gruenwald, and Obermeier (2006) discuss a protocol that aims at reducing aborts and blocking time. 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 (Mohan, Lindsay, & Obermark, 1986) and (b) the strict 2-phase-locking protocol (Eswaran, Gray, Lorie, & Traiger, 1976).

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 (see Figure 1). 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

Figure 1. State diagram for a master completing a transaction under the a 2-phase commit protocol



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 the 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*.

2. 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.

2.1 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.

In our modeling, an AmI environment is a set of cooperating nodes N. We deal with AmI environments as logical-level constructs that group nodes in a workgroup where they cooperate towards achieving a common goal. Depending on the case, this workgroup can be mapped to physical-level facilities (e.g., the AmI environment can be defined with respect to an area bounded by the connectivity range of a conventional networking infrastructure --e.g., a typical IEEE 802.11 network). More nodes can later join the environment and existing nodes can leave the environment. In the context of this paper, the terms 'join' and 'leave' the environment refer to a logical level participation to the AmI environment and not a physical one; in fact, the particularities of these actions at the physical level are orthogonal to the proposed protocol.

Our overall system model consists of a set of distinct AmI environments. Communication between nodes of N_i, N_j, for all $i, j \mid i \neq j$ is not possible.

The formation of the workgroup can be done by gathering the mobile nodes around a static, fixed point of reference (e.g., a standard access point in a building), or by arranging an ad-hoc network of mobile peers. In both of these cases, two kinds of nodes participate: (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 Figure 2, 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 such 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 (still, this particular assumption can be relaxed with straightforward enhancements in the proposed protocol.).

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). 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.

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.

Figure 2. Exemplary schedule of a mobile node

N_1	20
N_2	30
N_1	40
N_3	44

2.2 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).

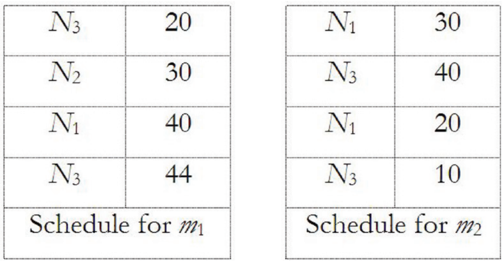
Assume a transaction that takes place in environment N_1 and involves a fixed master and two mobile cohorts, m_1 and m_2. Assume that at time point τ the master receives a message from cohort m_1 that the latter is leaving environment N_1. The schedules of the two cohorts at time point τ are depicted in Figure 3. The master, can calculate that, according to the cohorts' schedules, cohort m_1 will be back at the 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.

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 de-freeze, 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.

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.

Figure 3. Schedules for mobile nodes at the time of departure of m_1



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	

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.

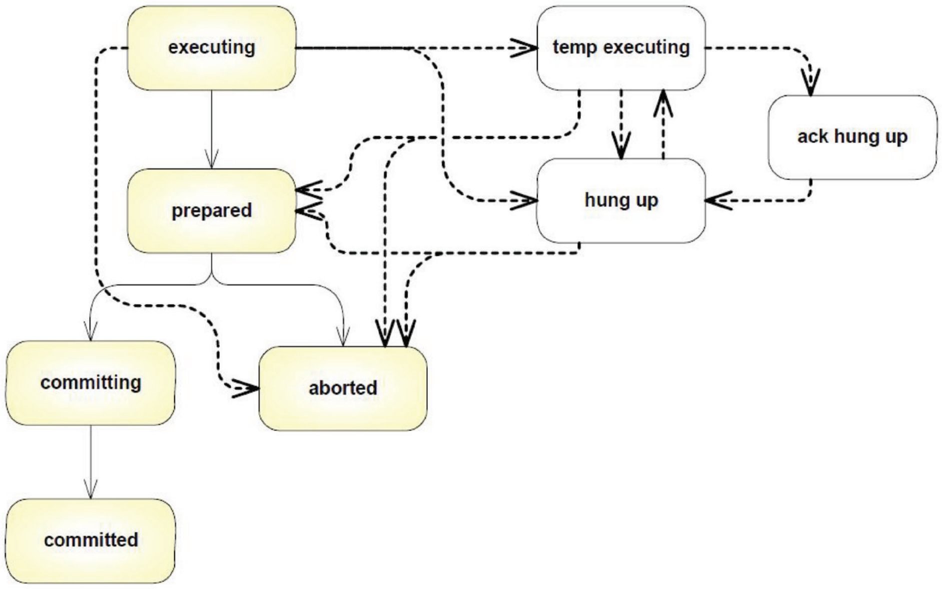
The execution of the transaction continues interactively. Whenever a participating cohort returns to the environment, the master node tries to execute the next step of the transaction. If the execution of the next step is possible the master passes in a *temp executing* state and keeps up with the execution 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.

Observe Figure 4 depicting the state diagram for the master in this case. The darker nodes correspond to the typical presume-abort 2-phase-commit protocol and the white nodes present the proposed extension.

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*

Figure 4. State diagram for the master, when a cohort leaves and the master is in prepared state



hung up state, waiting for the remaining cohorts' votes. When the master can reach a decision for the transaction, there are two cases:

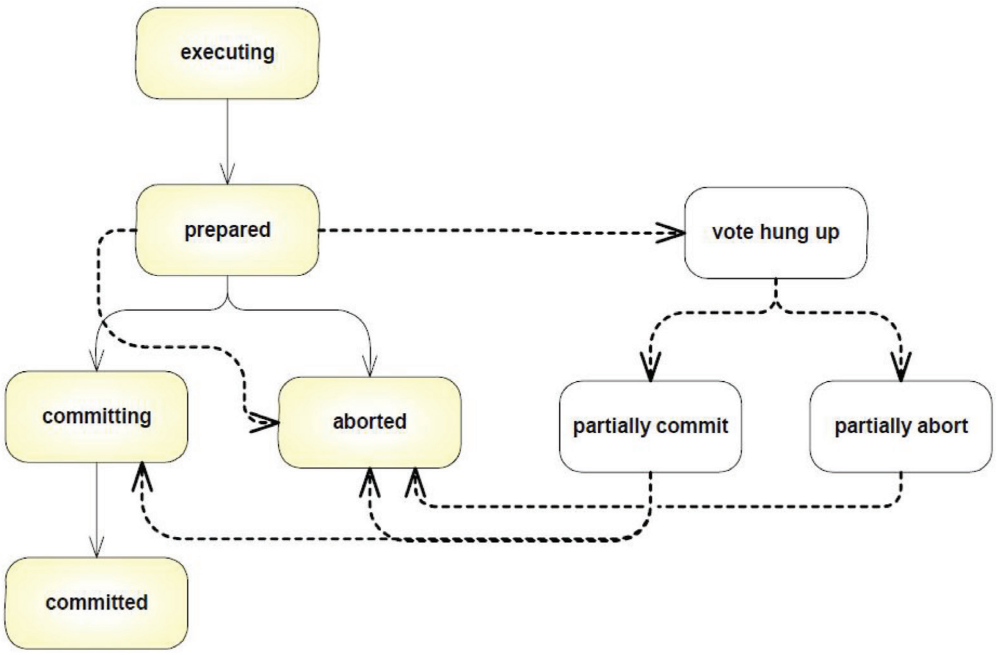
- If the transaction is to be aborted, the master notifies all cohorts that are present about the decision and assumes a *partially abort* state, until the rendezvous point. At this point, the master sends an *ABORT* message to the returning cohorts and moves to an *aborted* state. Note that some cohorts 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.

Observe Figure 5 depicting the state diagram for the master in this case. The darker nodes correspond to the typical presume-abort 2-phase-commit protocol and the white nodes present the proposed extension.

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

Figure 5. State diagram for the master, when a cohort leaves and the master is in prepared state



organize its departure and calculate the rendezvous, the master arranges *to send a LEAVE message to itself somewhat earlier than its departure*.

2.3 Discussion: Risks and Opportunities of the FOL Protocol

In this section we discuss possible risks and opportunities for improvement of the proposed protocol and explain some of our design choices.

Security and Privacy. A clear concern for the proposed protocol has to do with the fact that the cohorts' schedules must be released to the master resulting in a breach of privacy for the cohorts. *We should make clear that the proposed protocol operates under the assumption that the master is trusted*. If the master is not trusted by even one of the cohorts, then clearly, the transaction execution falls back to a schedule-agnostic mode. Also, it is not necessary to submit the full agenda of a cohort to the master; it is only sufficient to release a reasonably small subset of it for the context of a transaction. This can be achieved via a negotiation step during the handshake phase between the master and the cohorts. It is also possible to devise further optimizations, such as the anonymization of sensitive parts and the disclosure only of the case where the cohort will be back in the master's environment. Exploring these posibilities is an issue orthogonal to the protocol *per se*, especially since all of them result in the identification (or not) of common interval during which all the cohorts will be present in the same environment. So, for simplicity, in our deliberations we assume the simplest case, where all the cohorts automatically release their agendas to the master.

Concerning security, the transmission of the schedule to the master can be encrypted and even locally stored in an encrypted scheme; still, this is a topic for the implementing middleware to resolve and falls outside the scope of the paper.

Strictness of schedules. What happens if a node does not stick to its schedule? This is a realistic question that has to be answered by the protocol. There are mainly two cases:

(i) A cohort is late in its rendezvous. In this case, the master initiates an abort message and the late cohort presumes by default that an abort will occur. A simple extension of the protocol can even give some extra time after the rendezvous as a buffering period for latecomers; in fact this buffering period can even be calculated at the determination of the rendezvous time point. Still, this is a simple engineering extension to the protocol without significant implications.

(ii) A cohort is early in its rendezvous. This is no problem per se, if the cohort intends to stick to its previous schedule for the rest of its tasks. At the same time, this also presents an opportunity if *all* cohorts arrive early. It is possible to devise schemes to take this case into consideration; in our case we considered this to be a rare case and opted for a simpler protocol.

Other possible directions involve the monitoring of cohorts progress with respect to their registered schedule and the adaptation of the rendezvous point. We believe that the protocol should stick to *local scope principle*, in the sense that each master should only be interested in what happens in its specialized purview without global coordination or monitoring back-stage activities. Still, it is possible that in specialized situations, this could be performed with significant gains of committed transactions –at the expense, of course, of simplicity.

Opportunities for improvements. It is possible for the skeptical reader to raise questions related to the assumptions made in this paper. A simple example involves the role of environments in the whole setting: for example, if two different environments are close in terms of wireless transmission, or, if they have direct connection of their fixed nodes, is it possible to take advantage of this fact and improve the protocol? So far, we have assumed that an environment is an area within which the mobile nodes can communicate with each other, so, strictly speaking, as long as there is network connectivity among the involved nodes we should still consider that they are in the same environment. Still, it is possible to consider situations where an environment is bounded by geographical and connectivity constraints. Mesh networks, each employing a dedicated gateway node could possibly be considered in such a scenario and a cooperative scheme between them could be devised. We consider this opportunity as a topic for future research.

A second possibility has to do with the mobility of the nodes. Improvements of the protocol could be explored for the case that the nodes move groupwise, in a ‘herd’ fashion (Musolesi & Mascolo, 2007) as well in cases of other mobility models derived based on real-world observations (e.g., Bittner, Raffel, & Scholz, 2005; Tian, Haehner, Becker, Stepanov, & Rothermel, 2002). In general, a restriction of our model is that the nodes must return to the initiating environment to complete the transaction. It is possible to think of schemes where the nodes complete the transaction in another environment. Nevertheless, adopting such an approach would require total, detailed knowledge of all the schedules (against the aforementioned comments for privacy issues) and the environments and would result in a *global scope rendezvous* protocol. For practical purposes of efficiency and simplicity, we believe that a *local*, or at best, a *limited horizon scope* must be adopted, in which the rendezvous are considered without total knowledge of the network structure or the nodes' schedules. In other words, there is a trade-off between network and schedule knowledge, protocol simplicity and speed vs. the percentage of committed transactions. This trade-off is a function of the extent of the horizon that should be considered and its intricacies suggest another topic for future research.

3. 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] with a uniform distribution). 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 uniformly in the range of [1, 20]. The number of actions performed on each step was uniformly distributed in the range [1, 3]. Each action had a probability of 0.5 to be performed on a variable that belonged to a mobile node. Given that for each action a node is randomly selected with a uniform distribution, the number of nodes involved in the transactions was bounded by the number of steps that constituted the transactions. 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.

Figure 6 summarizes the results we obtained. More specifically, Figure 6 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 (see Figure 7), we have also measured the effect of the number of mobile nodes and the composition of transactions with respect

Figure 6. FOL vs. a schedule-agnostic protocol: Percentage of aborted transactions

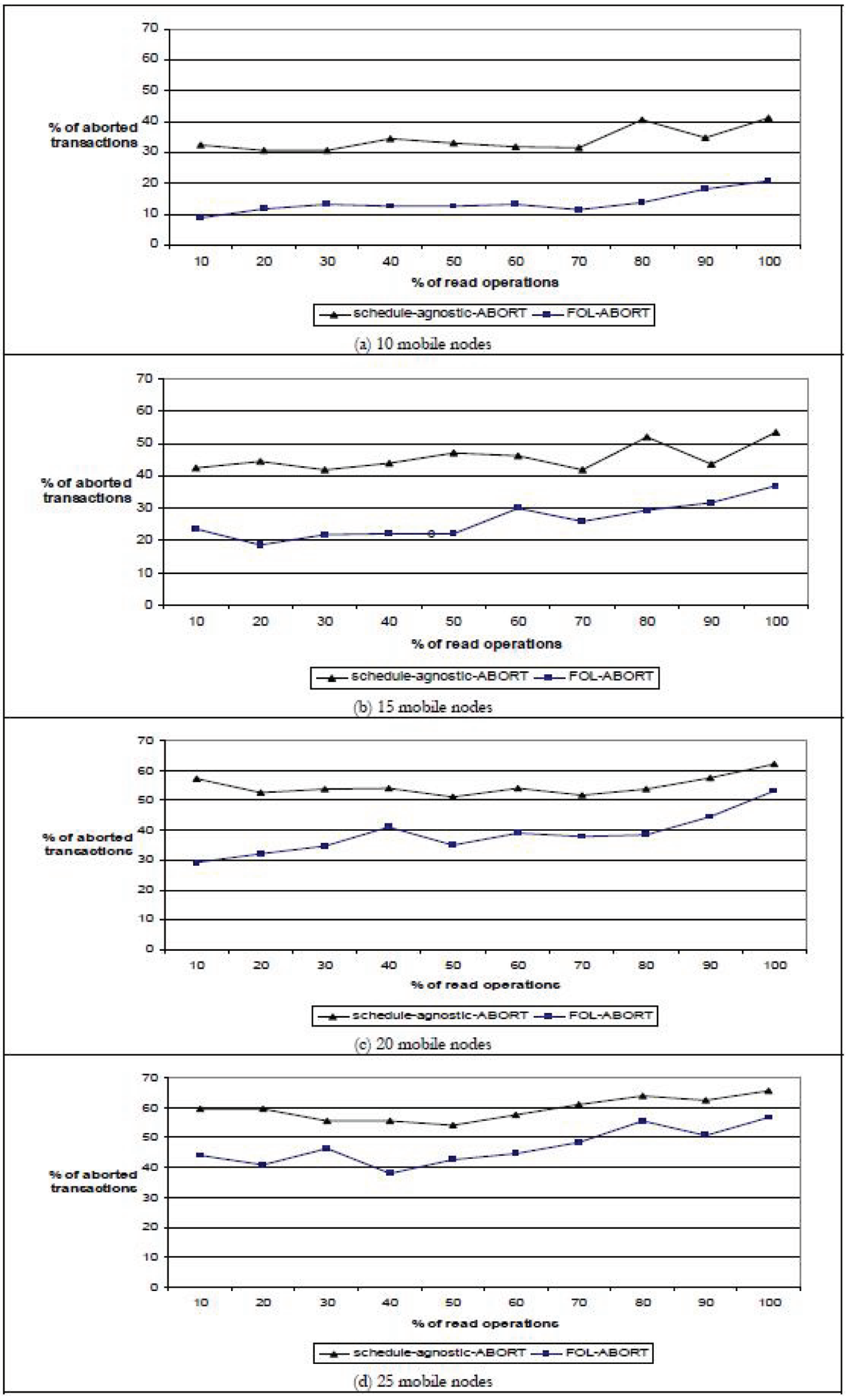
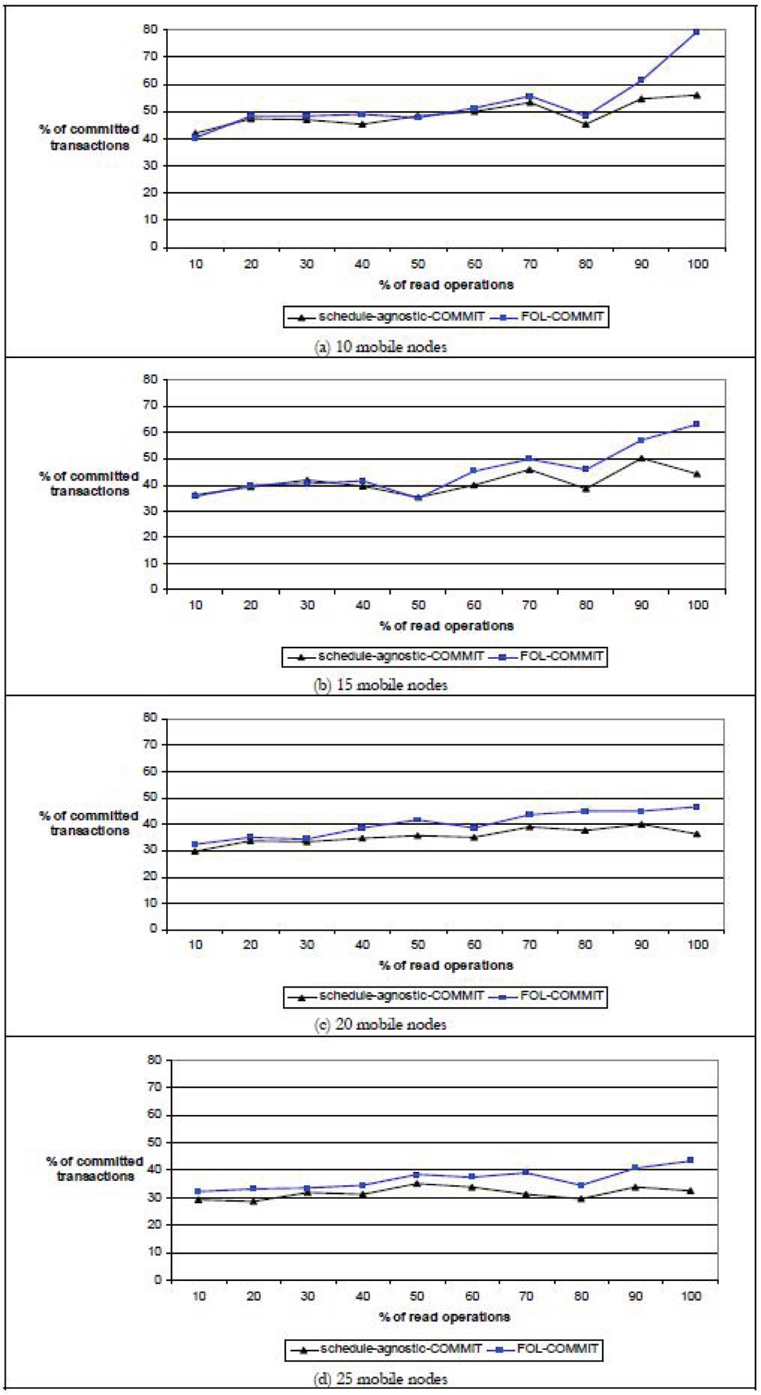


Figure 7. FOL vs. a schedule-agnostic protocol: Percentage of committed transactions



to reads and writes. The percentage of committed transactions is not the complement of the percentage of aborted transactions: this is due to the fact that due to strict locking of resources, starvations occur and some transactions never start. In this case we consider the transaction cancelled. If too many write operations take place, then the possibilities for concurrencies are reduced and many cancellations take place. Moreover, due to this fact, the schedule-aware and the schedule-agnostic protocols behave similarly. Still, as the number of read operations increases, more transactions can operate concurrently; the experiments show that (a) cancellations decrease (and the percentage of committed transactions increases) and (b) the schedule-aware protocol performs better than the schedule-agnostic one. Moreover, the difference between the two protocols becomes clearer as we increase the number of mobile nodes involved in the environments.

4. CONCLUSION 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 (e.g., probabilistic schedules, or schedules based on fuzzy sets). 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, which consequently involves several issues including the specification of interoperable schedules and monitoring the availability of nodes in a particular environment.

ACKNOWLEDGMENT

We would like to thank the reviewers of earlier versions of this paper for their valuable comments that significantly improved the clarity of the paper.

REFERENCES

Aarts, E., Harwig, R., & Schuurmans, M. (2001). The Invisible Future: The Seamless Integration of Technology into Everyday Life . In Denning, P. J. (Ed.), *Ambient Intelligence* (pp. 235–250). New York: McGraw-Hill.

Bittner, S., Raffel, W.-U., & Scholz, M. (2005, March 8-12). The Area Graph-based Mobility Model and its Impact on Data Dissemination. In *Proceedings of the PerCom 2005 Workshops, the 3rd International Conference on Pervasive Computing and Communications*, Kauai Island, HI (pp. 268-272).

Bobineau, C., Pucheral, P., & Abdallah, M. (2000). A Unilateral Commit Protocol for Mobile and Disconnected Computing. In *Proceedings of the 12th International Conference on Parallel and Distributed Computing Systems (PDCS'00)*, Las Vegas, NV.

Böttcher, S., Gruenwald, L., & Obermeier, S. (2006, July 18-20). Reducing Sub-transaction Aborts and Blocking Time within Atomic Commit Protocols. In *Proceedings of the 23rd British National Conference on Databases (BNCOD 23)*, Belfast, Northern Ireland, UK.

Eswaran, K. P., Gray, J. N., Lorie, R. A., & Traiger, I. L. (1976). The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, *19*(11), 624–633. doi:10.1145/360363.360369

Kumar, V., Prabhu, N., Dunham, M. H., & Seydim, A. Y. (2002). TCOT- A Timeout-based Mobile Transaction Commitment Protocol. *IEEE Transactions on Computers*, *51*(10), 1212–1218. doi:10.1109/TC.2002.1039846

Le, H. N., & Nygard, M. (2005, August 22-26). Mobile Transaction System for Supporting Mobile Work. In *Proceedings of the 16th IEEE International Workshop on Database and Expert Systems Applications, DEXA Workshops 2005*, Copenhagen, Denmark (pp. 1090-1094). Washington, DC: IEEE Computer Society.

Mohan, C., Lindsay, B., & Obermarck, R. (1986). Transaction Management in the R Distributed Database Management System. *ACM Transactions on Database Systems*, *11*(4), 378–396. doi:10.1145/7239.7266

Musolesi, M., & Mascolo, C. (2007). Designing Mobility Models Based on Social Network Theory. *ACM SIGMOBILE Mobile Computing and Communications Review*, *11*(3), 59–70. doi:10.1145/1317425.1317433

Nouali, N., Doucet, A., & Drias, H. (2005). A Two-Phase Commit Protocol for Mobile Wireless Environment. In *Proceedings of the Sixteenth Australasian Database Conference, Database Technologies 2005*, Newcastle, Australia (pp. 135-143).

Pitoura, E., & Samaras, G. (1998). *Data Management for Mobile Computing*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Serrano-Alvarado, P., Roncancio, C., & Adiba, M. (2004). A Survey of Mobile Transactions. *Distributed and Parallel Databases*, *16*(2), 193–230. doi:10.1023/B:DAPD.0000028552.69032.f9

Tian, J., Haehner, J., Becker, C., Stepanov, I., & Rothermel, K. (2002). Graph-based Mobility Model for Mobile Ad Hoc Network Simulation. In *Proceedings of the 35th Annual Simulation Symposium Annual Simulation Symposium,* San Diego, CA (pp. 337-344). Washington, DC: IEEE Computer Society.

Weber, W., Braun, C., Glaser, R., Gsottberger, Y., Halik, M., Jung, S., et al. (2003). Ambient Intelligence - Key Technologies in the Information Age. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM'03)* (pp. 1.1.1-1.1.8).

Weiser, M. (1991). The Computer of the Twenty-First Century. *Scientific American*, *135*, 94–104. doi:10.1038/scientificamerican0991-94

Younas, M., Chao, K.-M., Wang, P., & Huang, C.-L. (2007). QoS-aware Mobile Service Transactions in a Wireless Environment. *Concurrency and Computation*, *19*(8), 1219–1236. doi:10.1002/cpe.1157

Vasileios Fotopoulos received his B.Sc. in Computer Science from the Department of Computer Science, University of Ioannina in 2005. He received his M.Sc. in Computer Science from the same department in 2008. His research interests include middleware and P2P networks.

Apostolos Zarras received his B.Sc. in Computer Science in 1994 from the Computer Science Department of the University of Crete. From the same department he received his M.Sc. in Distributed Systems and Computer Architecture. In 1999 he received his Ph.D. in Distributed Systems and Software Architecture from the University of Rennes I. Now he holds an Assistant Professor position at the Department of Computer Science of the University of Ioannina and he is a member of the Distributed Management of Data (DMOD) Laboratory (http://www.dmod.cs.uoi.gr/). His research interests include middleware, model-driven architecture development, quality analysis of software systems and pervasive computing. Further information can be found at http://www.cs.uoi.gr/~zarras.

Panos Vassiliadis received his Ph.D. from the National Technical University of Athens in 2000. He joined the Department of Computer Science of the University of Ioannina as a lecturer in 2002. Currently, Dr. Vassiliadis is also a member of the Distributed Management of Data (DMOD) Laboratory (http://www.dmod.cs.uoi.gr/). His research interests include data warehousing, web services and database design and modeling. More information is available at http://www.cs.uoi.gr/~pvassil.