Pers Ubiquit Comput (2011) 15:821–831
DOI 10.1007/s00779-011-0369-1

ORIGINAL ARTICLE

A new model for context-aware transactions in mobile services

Muhammad Younas · Soraya Kouadri Mostéfaoui

Received: 10 December 2010 / Accepted: 5 March 2011 / Published online: 19 April 2011

Abstract With the ubiquity of handheld devices (such as smart phones and PDAs) and the availability of a wide range of mobile services (such as mobile banking, road traffic updates, and weather forecast), people can nowadays access information and conduct online transactions virtually anywhere and anytime. In such flexible, dynamic but less reliable environment, transaction management technology is believed to provide service reliability and data consistency. Indeed, in mobile and ubiquitous environments where devices as well as services can seamlessly join and leave the ubiquitous network; transaction management can be very helpful during the recovery of services from failure. Current transaction models and commit protocols do not take into account context information. However, in mobile environments, it is imperative to consider context information in the commit of a transaction—i.e., a transaction can be successfully completed if it meets the required context. In this paper, we propose a new model for context-aware transactions and their performance management in mobile environments. Unlike conventional transactions, context-aware transactions adapt to the required context. By context, we mean the service's context as well as the users' context that includes users' needs and preferences. This paper designs and develops the proposed transaction model and evaluates its performance in terms of time and message complexities as well as transaction's throughput.

Keywords Context-aware transaction · Mobile and ubiquitous services · Transaction model · Performance · Transaction properties

M. Younas (✉)
Department of Computing and Electronics,
Oxford Brookes University, Oxford, UK
e-mail: m.younas@brookes.ac.uk

S. Kouadri Mostéfaoui
Department of Communication and Systems,
The Open University, Milton Keynes, UK
e-mail: s.kouadri@open.ac.uk

1 Introduction

Recent developments in mobile technologies, such as wireless communication networks, handheld devices, and service standards enable people to access information and acquire services in a ubiquitous manner. For instance, Google Mobile provides users with access to a variety of services from their mobile phones, ranging from simple web page's browsing through to products' prices to driving directions. The service-oriented computing and web services technologies further facilitate the provision and consumption of mobile services [1]—i.e., find, select, and develop (or compose) new services from existing ones in order to provide enhanced functionalities, enable universal accessibility, and reduce operational and development costs through service reuse [2]. In the following discussion, we use the term mobile or M-services (as in [3, 4]) to refer to web services in the mobile environment.

The work presented in this paper investigates into the issue of transaction management in M-services. A transaction represents an abstract view of a sequence of operations that are involved in the execution of an application. Transaction management has been used in a number of applications such as databases, e-commerce, m-commerce, and engineering applications [5, 6]. We believe that transaction management is important for M-services, since protecting and managing the integrity of M-services'

outcomes is essential in a ubiquitous environments. Specifically, from the users' perspective, transaction management should guarantee that services obtained using mobile devices are consistent with what users request and what service providers offer. From the M-service providers' perspective, transaction management must ensure that transactions are correctly executed, the enterprise has correct information about the outcomes of those transactions, and information held in an enterprise's databases is maintained to provide a truthful and consistent record of the state of the enterprises.

Various models and protocols have been developed for mobile transactions. However, they are limited to the classical commit procedures that do not give attention to context-awareness. Consider an example where Sofia is using her iPhone while walking in the Broad Street in Oxford, UK. Sofia intends to book a table in a nearby Italian restaurant for a dinner with her friends. Using her iPhone, she browses mobile services in order to find appropriate restaurants, check the location of nearby restaurants, and also check the menus and prices of those restaurants. Current approaches will simply commit a transaction whether the required table is available in a restaurant. They do not take into account the "context" information such as "a table should be booked in a restaurant which is located nearby". Existing literature contains significant work on context-aware systems [7–10]. But to our knowledge, context awareness is not addressed in the transaction management in general and mobile transaction management in particular. The philosophy of context awareness is that systems must automatically adapt their operations to the environment by taking into account context information such as current location, time, users' needs, and other environmental parameters. Thus, in M-services, it is desirable that a transaction reacts to the contextual information and adapts its behavior according to the changes in context. Furthermore, current approaches are generally based on the conventional ACID (atomicity, consistency, isolation, and durability) criteria or ETMs (extended transaction models) [11]. ACID criteria require that a transaction must not expose its intermediate results (isolation) and must be atomic (all its actions must be carried out or none is). ETMs relax the ACID criteria especially the isolation property. However, neither the ACID nor the ETMs take into account the context in transaction management.

The contributions of this paper are defined as follows.

- *To develop a new model for context-aware transactions in mobile services environments.*
  This paper proposes a new set of transaction correctness criteria, called RACCD (relaxed atomicity, consistency, context, and durability). RACCD criteria include "context" as one of the main properties of transactions, which has not been considered in the existing transaction models. The new criteria are believed to be more appropriate to the context aware, more volatile, dynamic, and open nature of transactions in M-services environment.
- *To design and develop a protocol for the execution of context-aware transactions.*
  We design and develop a protocol in order to enforce the RACCD criteria during the execution of context-aware transactions. The protocol is developed in multiphases in order to collect service context, reserve required services, and acquire the required services by successfully completing (committing) a transaction.
- *To implement the proposed model and evaluate its performance.*
  We implement the proposed model as a prototype system, called "Smart Campus". Analytical model is also developed in order to evaluate the performance of the proposed model in terms of message and time complexities and transaction's throughput

The remainder of the paper is organized as follows. Section 2 describes the basic definitions of M-services and analyses their characteristics. Section 3 reviews the related work and establishes its limitations with respect to the requirements of M-services transaction management. Section 4 presents the proposed context-aware transaction model. System architecture and the execution protocol are described in Sect. 5. Section 6 describes the evaluation and validation of the proposed model. Finally, the paper is concluded and directions for future work are identified in Sect. 7.

2 Background

2.1 M-services: basic concepts

M-services are software applications that represent a higher level abstraction of a set of activities that manipulate different resources in order to fulfill users' requests [2–4]. The proposed model requires that M-services are developed using the service-oriented computing (SOC) architecture and technologies as they enable dynamic composition of loosely coupled distributed services in order to facilitate software reuse, provide enhanced functionality, reduce operational and development costs, and to enable inter-organizational collaboration. The most common implementation technologies of SOC are the XML web services, which are built using WSDL, SOAP, REST, and UDDI among others. M-services are published using service interface definition languages, WSDL, or CWSDL [12, 13],

such that they can be consumed (invoked) by service consumer applications (or M-services transactions in our case).

2.2 Context

Based on an instantiation and combination of Dey's and Schilit's definitions [14, 15], context is defined as "any information that can be used to describe the situation of people, resources, and services in a service-oriented environment. It may include all other information that can be considered relevant to the interaction between a user and a service". Context information of M-services can be obtained into two different ways:

a. *Internal context* It represents context information related to the service that can be directly obtained from the service interface definition, for example, using the Context-based Web Service Description Language (CWSDL) [13]. CWSDL was developed in order to enhance the standard W3C WSDL and to provide support for context-related information. CWSDL defines the service profiles and the related quality metrics for rating, comparing, and selecting the desired services.
b. *External context* It represents any context information which is external to the service that can be obtained using external (independent) services. For example, location service can provide location information about a hotel service, and a weather service can provide weather forecast about a particular location.

2.3 M-services characteristics

A generalized architecture of mobile systems (in Fig. 1) is presented in order to illustrate the characteristics of M-services [16]. In such architecture, M-services may reside on a fixed host (FH) or on a mobile host such as a mobile device. Base stations (BSs), controlled by a base station controllers (BSCs), are capable of communicating with mobile devices through wireless networks. BSCs are in turn controlled by the mobile switching center (MSC) that is connected to the Internet. Each BS covers a particular area, called a cell. Handoff happens when a mobile device moves from one cell to another.

Transactions in M-services have different characteristics than those of static services. In M-services, transactions are less prescribed, more prone to failure, and open-ended. Also characteristics specific to mobile devices and networks add further complexity. Specifically, the mobility of the processing units (e.g., PDA) means transactions may relocate during execution, as their originating devices

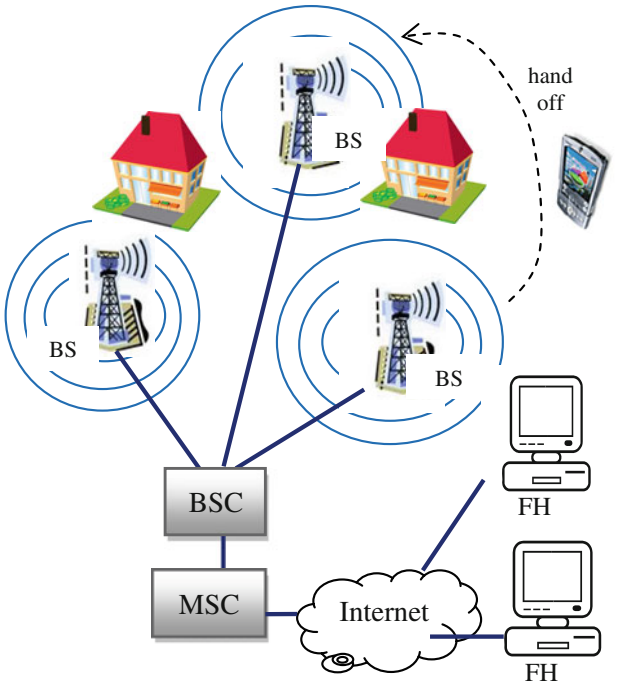

Fig. 1 Generalized architecture of mobile systems

move. Also, connection with the mobile devices may be intermittent. Thus, for example, a transaction may originate at one site and terminate at another and require frequent disconnection and reconnections in between. Also, the bandwidth of mobile networks is currently low in comparison to wired networks. Consequently, communication links may be overloaded by high volume of information exchange. In addition, mobile processing units are currently less reliable and with fewer resources, than the conventional "stationary" units. For example, the limited power supplies of mobile devices and the relatively limited resources affect failure resilience of M-services, since they increase the probability of transaction failure, for example, due to a flat battery or running out of memory.

In addition to the above, the issue of context awareness further complicates the management of transactions in M-services. In a restaurant booking example, a context-aware transaction will be considered as failed (aborted) transaction if it books a table in a restaurant that is outside the specified location. In this case, the service is available but it does not meet the desired context. However, with current transaction management approaches, a transaction will be successfully processed (committed) if it can book the table in a restaurant irrespective of the location. Another example of context-related failure can be a performance failure, that is, a transaction will fail if it does not respond within a specified time interval.

3 Literature review

There is a scarcity of literature on context-aware transactions in M-services. This section, therefore, reviews only transaction management techniques that are developed for mobile database transactions but can be considered for M-services transactions. Holanda et al. [17] develop an intelligent transaction scheduler using a combination of conservative and aggressive concurrency control protocols. The proposed scheduler is claimed to be context aware in the sense that it automatically identifies changes in the computational environment and adapts to the appropriate concurrency control protocol. The efficiency of the scheduler is tested using the mobile database community that represents a set of databases connected through MANET. Rouvoy et al. [18] propose Context-Aware Transaction sErvice (CATE) that provides facilities for selecting appropriate protocol (among the conventional 2PC, PC, and PA protocols) that meets the execution context. CATE is claimed to improve performance as compared to using only one commit protocol. The context is defined in terms of the commit and abort rates of a transaction. That is, based on the number of commit and abort transactions, an appropriate commit protocol is chosen for the processing of a transaction. However, the above approaches [17, 18] are limited to the classical concurrency and commit protocols and do not take into account the context information such as location, time, and so on.

Kumar et al. [19] define a protocol, called TCOT, for mobile transactions. TCOT, a variant of the classical two-phase commit (2PC) protocol, is based on a timeout mechanism. TCOT is claimed to have improved performance and throughput over 2PC protocol in managing mobile transactions. 2PC protocol enforces classical ACID criteria [6]. As described above, 2PC and ACID criteria are inappropriate for M-services transactions. Lee et al. [20] introduce a mobile transaction model called high commit mobile transactions (HiCoMo) in order to improve the commitment rate of mobile transactions. Using HiCoMo transactions, aggregate data can be updated in a situation when mobile units are disconnected. The Kangaroo transaction (KT) model [21] takes into account the movement behavior of mobile transactions such that they can hop from one base station to another as their mobile device moves. But in certain situations, it may not help in reducing communication overhead. In summary, the above approaches are limited to the classical mobile transactions and they do not consider M-services or context awareness. The authors in [22] proposed an adaptive context transaction model for mobile and ubiquitous services. This work is interesting and it develops performance management model. However, unlike our approach, this work does not consider "context" as a first class correctness property. Furthermore, WS-Transactions [23] aim to address issues related to 2PC-based protocols. WS-Transactions define a framework for providing transactional coordination of web services offered by multiple autonomous businesses.

4 The context-aware transaction model

This section first illustrates the definitions of context-aware transactions. It then presents the new properties defined for the context-aware transactions.

4.1 Context-aware transactions

Context-aware transactions can range over different M-services involving and spanning many enterprises distributed over the wired and wireless networks. In the proposed model, a context-aware transaction, denoted CA_T, is defined as an execution of a composite service which can be divided into component service transactions, denoted cst_i. The execution of CA_T aims to acquire M-services (required by a user) and to maintain the correctness and consistency of those services.

Formally, CA_T is defined as a tuple, $CA_T = (MS_i, cst_i, <)$; where MS_i $(1 \leq i \geq n)$ represents M-services such as a restaurant service or a location service. The cst_i $(1 \leq i \geq n)$ is a set of component services transactions. Each of the cst_i is a sequence of operations that are executed in order to collect service context, reserve services, and acquire (or commit) services (MS_i) $<$ is a partial ordering of the cst_i, which determines the order of execution of cst_i. The cst_i may have different types such as compensatable, vital, and non-vital. The cst_i is compensatable if its effects can be semantically undone by executing a compensating action. All vital cst_i must be successfully executed in order for the CA_T to commit. If any of the vital cst_i fails, then the CA_T will be aborted. Failure of non-vital cst_i may not result in the abort of the CA_T. Consider an $CA_T$for a restaurant booking which composes different component service transactions, $CA_T = \{cst_1, cst_2, cst_3, cst_4\}$—where cst_1 checks the location of a restaurant, cst_2 chooses a restaurant and reserves a table, cst_3 makes payment for table's reservation, and cst_4 gives the directions to a restaurant. In this hypothetical transaction, if a user can get the restaurant location, book a table and make payment, then the transaction will be considered as successful even if she does not get the directions to the restaurant. Thus, the component service transactions cst_1, cst_2, cst_3 are considered as vital, while cst_4 is non-vital as it does not result in the failure (abortion) of CA_T even if it's not successfully completed. Furthermore, we assume a flat structure of CA_T; meaning that it can be composed of single level cst_i. However, a

nested or hierarchical CA_T can be constructed based on the flat structure.

The characteristics of context-aware transactions are significantly different from the classical transactions. Context-aware transactions:

- involve different M-services, which are distributed across the (wireless and wired) networks and are provided by a number of independent and autonomous service providers
- need to fulfill the contextual requirement of the required services
- may suffer from a variety of failures such as system failures and context-related failures
- need longer processing time, e.g., gathering of context information and acquiring M-services.

The above characteristics reveal that current transaction models and properties (such as ACID criteria or ETMs) are inapplicable to the context-aware transactions. These motivate the need for a new transaction model and correctness properties, which are presented below.

4.2 Context-aware transaction properties

We propose new criteria, called RACCD (relaxed atomicity, consistency, context, and durability). Each context-aware transaction is required to meet the rules set by these criteria:

- *Relaxed Atomicity (RA)* It relaxes the strict notion of the classical atomicity which requires that either all or none of the operations (in our case, cst_i) of an CA_T be completed. In M-services environment, such restriction is inappropriate due to the nature of independent and autonomous services. The relaxed atomicity allows partial commit of CA_T such that individual cst_i may commit unilaterally and the failure of non-vital cst_i may not result in the abortion of the CA_T. Relaxed atomicity requires that all of the (vital) cst_i must be successfully executed in order for CA_T to be committed. In the above example, either all or none of the vital cst_1, cst_2, and cst_3 is executed in order for CA_T to maintain the RA property. Since RA allows unilateral commit, it is possible that some cst_i successfully execute while others fail. In that case, it is required to compensate the effects of the executed cst_i via compensation actions in order to maintain the RA property.
- *Consistency* Consistency requires that the M-services data remain consistent after the execution of CA_T. The traditional notion of consistency and isolation of ACID criteria cannot be enforced in M-services as they either result in the useless consumption of resources or resource blocking. That is, they require all the component services to wait for the completion of each other in the prepare-to-commit state [6].
- *Context* It requires that CA_T must fulfill the requirements of the service context. That is, CA_T can be committed only if all its vital cst_i are successfully completed and they meet the required context of M-services.
- *Durability* Similar to the ACID criteria, it requires that effects of a committed CA_T must be made permanent in the respective data sources of M-services, even in the case of failures.

5 Architecture and protocol

5.1 The system architecture

This section presents a generalized system architecture within which the context-aware transaction protocol is implemented. The system architecture is shown in Fig. 2. Users submit context-aware transactions (CA_T) to the system implementing the *main coordinator* (MC). MC manages the overall processing and execution of a CA_T. MC submits the component services transactions (cst_i) to the *component coordinators* (CC_i), which are associated with various *M-services* (MS_i) such as a restaurant service or a weather service. Each CC_i executes cst_i on MS_i in order to collect context information, reserve, and acquire services. Referring to Fig. 1, we assume that MC and CC can be deployed at fixed hosts where M-services may be deployed at fixed as well as mobile hosts. MC and CC are assumed to be deployed at fixed hosts as they are considered to be more reliable than mobile hosts. Further, it is assumed that each CA_T contains a finite number of cst_i and the type of cst_i (vital, non-vital) is specified by a user.

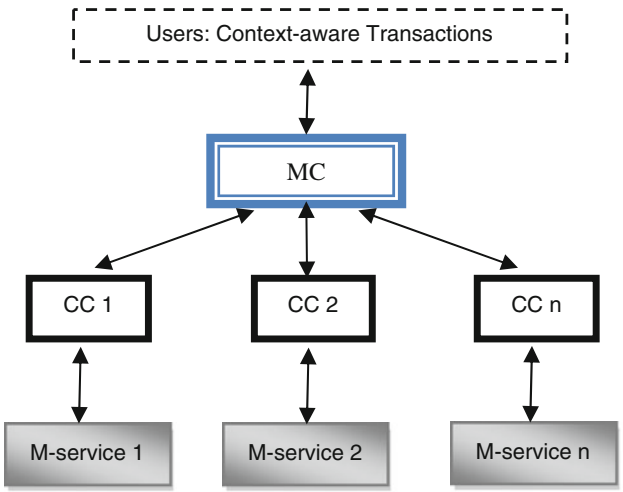

Fig. 2 The proposed system architecture

M-services transactions are coordinated by the MC which ensures that they comply with the rules set by the RACCD criteria. MC also maintains the execution order between $cst_i \in CA_T$. For instance, a compensating action cannot execute before the completion of an actual cst_i. That is, a cst_i for booking a table in a restaurant cannot be compensated for whether the table was not booked. MC also sends the type of each cst_i to the respective CC. Each CC executes cst_i on the M-service and sends the result to the MC.

MC and each CC are required to gather context information and also reserve services required by context-aware transactions. Service reservation is required in order to avoid blocking of services. As described earlier, context-aware transactions are generally of longer duration, and thus, it is not feasible to use the classical locking protocols such as two-phase locking [11]—where a service can be locked by a single transaction and no other transactions can access that service (e.g., a table in a restaurant) until the lock is released. In the following, we adopt the W3C Tentative Hold Protocol (THP) [24] that allows multiple transactions to tentatively reserve the same service at the same time. For instance, the same table in a restaurant can be simultaneously reserved by two transactions, say, $cst_i \in CA_{T1}$, and $cst_j \in CA_{T2}$. However, if CA_{T1} completes before CA_{T2}, then the reservation of cst_j ($\in CA_2$) is canceled. Though THP may degrade the performance of context-aware transactions, it provides more flexibility as multiple transactions are able to request tentative reservation of the required services, check their availability as well as their context. It also provides flexibility to service providers as they can allow multiple non-blocking reservation of their services, avoid useless consumption of their services (which may happen through classical locking) thus retaining control of their resources. Most importantly, THP increases the throughput (commit rate) of transactions and thus reducing the need to execute compensating actions.

As part of the THP, the coordinators (MC and CC) implement the M-service policy framework. As in THP [24], the policy framework implements the rules set by the service provider. These rules define various conditions/restrictions on the tentative reservation of services, for example, setting a timeout interval for a tentative reservation, the number of tentative holds (reservation) on a particular service, and so on.

5.2 Context-aware transaction protocol

In the first phase, component service transactions are executed in order to collect the service context information and reserve the required M-services. The second phase executes the commit process for the context-aware transactions.

5.2.1 *Phase 1: context gathering and service reservation*

In this phase, the main coordinator, MC and each component coordinators, CC, implement the context gathering and service reservation protocol. The objective is to enforce the property of "context" in context-aware transactions and to avoid the blocking of resources (or data) as in classical 2PC protocols.

1.1 Users submit context-aware transaction, CA_T, to the system. After initiating the transaction process, MC logs the start of CA_T in a log file in order to keep the necessary information about CA_T and its component services transactions, cst_i, including transaction state information, recovery information, the execution order, and so on.

1.2 After logging the information, MC starts collecting context information. For example, if CA_T has to book a restaurant, then it needs to collect context information about the restaurant service. MC contacts each CC associated with the service for which the context is required. Each CC processes the request as follows.

   a. Upon receiving the request, CC starts collecting the context information by executing cst_i. For the sake of simplicity, CC is designed such that it is capable of collecting both internal as well as external context of the required service. Note that for every service context, a listener is attached that informs the CC whenever there is a change in context. CC collects:

      i. *internal context* information from a service interface definition, e.g., using CSWDL (as described above) or
      ii. *external context* from a separate independent service (e.g., location service).

   b. After executing cst_i, CC sends the required context information to the MC—(e.g., sending the restaurant location information).

1.3 MC checks (with feedback from users) the context information received from CC. If the information received does not meet the desired context, then MC has to cancel the CA_T as it cannot preserve the property of "context".

1.4 If the desired context is met, then MC sends a reservation request to CC. Service reservation is done using the THP protocol.
Upon receiving the request, CC executes cst_i in order to reserve the required service. The reservation process works as follows.

   a. CC executes cst_i to tentatively reserve the service. It uses the M-service policy framework

to determine whether the required service can be tentatively reserved. Note that tentative reservation must comply with the rules set by the M-service policy framework such as timeout interval for a tentative reservation and the number of tentative holds (reservation) on a particular service, etc.

b. If the required service can be reserved, CC sends a "res-yes" message to the MC telling it that the service is tentatively reserved. If not, CC sends "res-no" to the MC.

1.5 MC checks the service reservation response and also records it in the log file.

a. If the response is "res-yes", MC then starts the commit process (phase 2) provided no more context information or service reservation is required. If required, then the above steps for context gathering and service reservation are repeated.

b. If the response is "res-no", MC checks the type of cst_i (associated with MS_i). If the type of cst_i is vital, then MC has to cancel CA_T as it cannot preserve the relaxed atomicity property. If the type is non-vital, then MC proceeds to the commit process provided no more context information or service reservation is required. If required, then the above steps for context gathering and service reservation are repeated.

5.2.2 *Phase 2: commit process*

2.1 Once the context information is gathered and services reserved, MC starts the commit process. MC contacts each CC to commit their respective component service transactions, cst_i.

2.2 Each CC starts processing its respective cst_i. If cst_i commits (i.e., locally committed), CC sends a commit message to MC. If cst_i does not commit, CC sends an abort message to MC.

2.3 MC checks the messages received from each CC regarding the commit/abort of the component service transactions, cst_i.

a. If all the messages received are "commit" then MC commits the CA_T. It informs each CC about the commit decision of CA_T. Each CC then marks its cst_i as globally committed. MC logs the commit decision of CA_T, terminates it and starts a new CA_T (if any).

b. If any of the messages received is "abort" and the type of cst_i is non-vital then MC acts according to the above step, 2.3 (a).

c. If any of the messages received is "abort" and the type of cst_i is vital, then MC has to abort CA_T as one of its vital component service transaction is not committed. CA_T is aborted in order to maintain relaxed atomicity, which enforces the condition that all the vital component service transactions must be committed.

i. MC sends abort message to each CC that has locally committed cst_i.

ii. Each CC executes compensating actions in order to cancel the effects of cst_i.

iii. After compensation, each CC marks its cst_i as aborted.

d. Following step (c), MC aborts CA_T and terminates it. MC then starts a new CA_T (if any).

6 Evaluation and validation

This section develops an analytical model for the performance evaluation of the proposed model and presents various experimental results. It also describes the implementation of a prototype system in order to validate the proposed model.

6.1 Performance evaluation

In this section, we evaluate the performance of the proposed protocol using analytical modeling technique. Performance is evaluated using the criteria of the time complexity, message complexity, and transaction throughput (commit rate). These criteria have also been used to evaluate the performance of the commonly used protocols such as two-phase commit, presumed abort, and the mobile transaction protocols [6, 19, 21]. Message complexity takes into account the number of messages required to process a context-aware transaction. The number of messages significantly affects the performance, as the time required to send a message between the participants of a protocol increases with the increase in the number of messages. Time complexity takes into account the processing time of component service transactions and also the disconnection delay. Disconnection delay is considered in the performance evaluation as context-aware transactions may suffer from disconnection due to the unreliable nature of mobile services environment (as discussed above). Disconnection affects the performance as it generally increases the execution time of context-aware transactions.

We make the following assumptions in the performance evaluation:

- Majority of the existing performance models for transaction protocols assume ideal conditions (failure free) where transactions do not suffer from failures such as communication failure, system failures, or software failure. We also assume such ideal conditions in the performance evaluation of the proposed protocol.
- Disconnections are not treated as failures that result in transaction's abort. It is assumed that during the execution of the protocol, component systems can be disconnected and reconnected (automatically) such that they do not result in the failure of context-aware transactions.
- Communication between the main coordinator and component coordinators (and services) is assumed to be done using wired (fixed) networks.

In the following, we develop a set of equations which are used to calculate the average processing time of a context-aware transaction. It is calculated using the performance parameters such as the execution time spent in context gathering, service reservation, transaction commitment, message communication, and the disconnection delay. The different parameters used in the performance evaluation are shown in Table 1.

Table 1 Parameters used in the performance evaluation

Parameter	Description
PT_{avg}	Processing time of a context-aware transaction, CA_T
CT_{commit}	Time taken in the commit phase of CA_T
EXE_{com}	Time taken to commit a component service transaction (cst)
T_{com}	Time taken in local processing of the commit of a cst
N_{cst}	Total number of component service transactions (cst). Each of the cst is used to gather context, reserve, and acquire (commit) services
P_{con}	Probability that a particular cst is used in gathering context
EXE_{con}	Time taken in gathering service context including message delay and local processing
T_{con}	Time taken in local processing of cst to gather context
EXE_{res}	Time taken to reserve a service including message delay and local processing
T_{res}	Time taken in local processing of cst to reserve services
C_T	Total time required to collect context information
R_T	Total time required to reserve the required services
W_{net}	Time to transmit a message over a wireless network
F_{net}	Time to transmit a message over a fixed (wired) network
DRC_t	Time spent in disconnection and reconnection
PDC_{con}	Probability of disconnection occurring during context gathering
PDC_{res}	Probability of disconnection occurring during service reservation
PDC_{com}	Probability of disconnection occurring during commit

The average processing time, PT_{avg}, is the time taken to collect the context information, reserve the services, and commit the transaction. PT_{avg} is calculated as follows.

$$PT_{avg} = C_T + R_T + CT_{commit} \quad (1)$$

We calculate the time taken to collect the context information, C_T, as follows.

$$C_T = EXE_{con} * (P_{con} * N_{cst}) + (PDC_{con} * DRC_t) \quad (2)$$

That is, the time taken in gathering context (EXE_{con}) times, the number of cst used in context gathering plus the disconnection delay with some probability. Note that for some of the services (e.g., payment service), cst may not be required to gather context. Hence, we use the parameter, P_{con}, which represents the probability of a cst used in gathering context.

Disconnection delay is included as it affects the processing time of a context-aware transaction. For simplicity, we assume that at most one disconnection happens during each of the phases of the context gathering, service reservation, and transaction commit. EXE_{con} is calculated as follows.

$$EXE_{con} = 3W_{net} + 2F_{net} + T_{con} \quad (2a)$$

According to 2a, the execution time, EXE_{con}, incurs:

- local processing time, T_{con}, in gathering context
- three messages over wireless network, W_{net}: one for MC (main coordinator) receiving a request (from user), one for MC sending the outcome (context information) to the user, and one for getting feedback from user
- two messages over wired network, F_{net}: one for MC to send context request to CC (component coordinator) and another for CC to send the context outcome to MC.

The time taken to reserve services, R_T, is calculated as follows.

$$R_T = (EXE_{res} * N_{cst}) + (PDC_{res} * DRC_t) \quad (3)$$

That is, the time taken to reserve services (EXE_{res}) times, the number of cst used in service reservation plus the disconnection delay. We assume that all services which are to be acquired by transactions are required to be reserved. EXE_{res} is calculated as follows.

$$EXE_{res} = 2F_{net} + T_{res} \quad (3a)$$

According to 3a, the execution time, EXE_{res}, incurs:

- local processing time, T_{res}, to reserve services
- two messages over wired network, F_{net}: one for MC to send service reservation request to CC (component coordinator) and another for CC to send the reservation outcome to MC. Note that reservation does not involve communication with (user's mobile device) and hence no wireless messages.

The time taken to commit a component service transaction, CT_{commit}, is calculated as follows.

$$CT_{commit} = (EXE_{com} * N_{cst}) + (PDC_{com} * DRC_t) \quad (4)$$

That is, the time taken to commit a cst (EXE_{com}) times, the number of cst to be committed plus the disconnection delay. EXE_{com} is calculated as follows.

$$EXE_{com} = 2F_{net} + W_{net} + T_{com} \quad (4a)$$

According to 4a, the execution time, EXE_{com}, incurs:

- local processing time, T_{com}, to commit a cst
- two messages over wired network, F_{net}: one for MC to send commit request to CC and one from CC to MC (to send the outcome of commit)
- one message, W_{net}, over wireless network to send the final result of a transaction to the user's mobile device.

6.2 Experimental results

In order to evaluate the performance of the proposed protocol, we have conducted various analytical experiments using the above equations (see Sect. 6.1). As context-aware transaction research is a new, values for many of the parameters cannot be known exactly. We obtain some of these values from existing work on (mobile) transactions, while for others we make educated guesses. For instance, the time to deliver a message over wired network (F_{net}) is 5 ms and over wireless network (W_{net}) is 10 ms (as in [19, 25]). However, the message communication time varies according to the type of network and also the network traffic. The average local processing time to commit (T_{com}) a cst is assumed to be between 50 and 90 ms (as in [6]). But the local processing time also varies as some transactions involve a large number of I/O operations and a greater deal of CPU than others. In the experiments, we use different values for all of the performance parameters used in the equations.

First, we compute the average processing time, PT_{avg}, of a context-aware transaction (CT_A) in three different cases. Each case varies according to the number of the component service transaction, N_{cst}, of CT_A. We set N_{cst} to 3, 4, and 5 cst in case 1, case 2, and case 3, respectively. The values (calculated in ms) for the other parameters are kept similar across these three cases, i.e., $P_{con} = 0.90$, $T_{com} = 70$, $T_{con} = 55$, $T_{res} = 45$, $W_{net} = 10$ $F_{net} = 5$, $PDC_{con} = PDC_{res} = PDC_{com} = 0.03$, and $DRC_t = 500$. Figure 3 shows the PT_{avg} for each of the three cases.

It is clear that a CT_A with a larger number of component service transactions (cst) needs more processing time than the one with small number of cst. In the performance evaluation of the classical commit protocols, it is assumed that component transactions are processed in parallel [11],

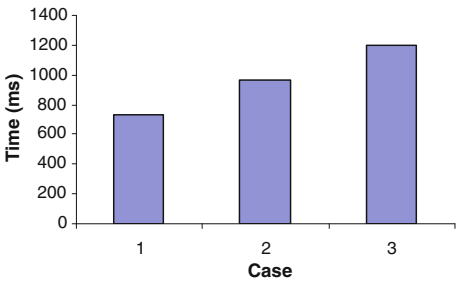

Fig. 3 Average processing time, PT_{avg}

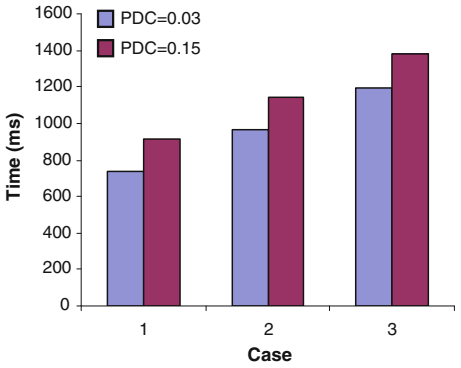

Fig. 4 Effects of disconnection on PT_{avg}

and hence, the increase in the number of component transaction does not affect the overall processing time of a transaction. However, in context-aware transactions, this assumption may not be valid due to the characteristics of M-services.

Next, we analyze the effects of the disconnection on the average processing of a context-aware transaction (Fig. 4). We use the above data but increase the probabilities of disconnection ($PDC_{con} = PDC_{res} = PDC_{com}$) from 0.03 to 0.15 in the different phases of the protocol. For the sake of simplicity, we assign the same values to the probabilities of disconnection during context gathering, service reservation, and commit phases. It is observed that disconnection significantly degrades the performance of context-aware service transactions and it must be minimized using appropriate methods.

The other main factor that affects the performance of context-aware transactions in the proposed protocol is the reservation of services.

Figure 5 shows two scenarios of the average processing time of a context-aware transaction—one with service reservation and one without service reservation. Though the service reservation increases the average processing time, it significantly improves the throughput (commit rate) of context-aware transactions (as shown in Fig. 6). It also reduces the chances of executing compensating actions in case of a transaction abort.

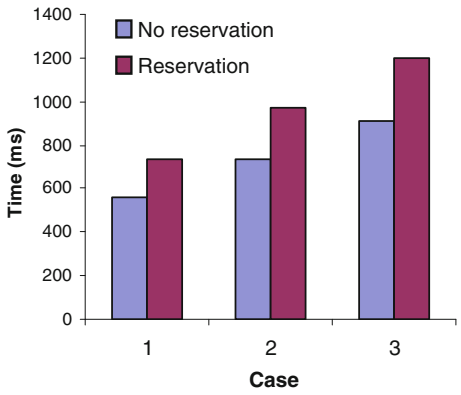

Fig. 5 Overhead of service reservation

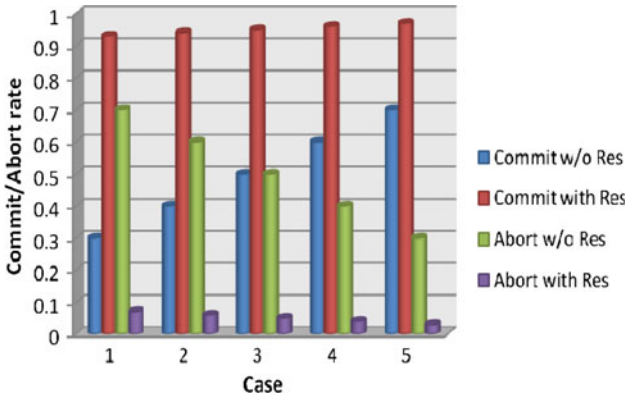

Fig. 6 Effects of service reservation on throughput

Next, we adopt the method from [12] in order to analyze the effect of service reservation on the throughput of a CA_T. This method is based on the probability theory which assigns various probabilities to the commit and abort actions of a CA_T. We conduct different experiments using various probabilities, and accordingly, calculate the throughput (commit rate) of CA_T as shown in Fig. 6. It shows that the commit rate of a CA_T significantly increases using the service reservation. Thus, the benefits of service reservation outweigh its limitation in terms of the processing overhead.

6.3 Prototype implementation

The proposed model is implemented as a prototype, called "Smart Campus" [26]. The implementation architecture is shown in Fig. 7. The aim of the Smart Campus prototype is to assist students and university staff with a mobile lightweight interface that provides them with context-aware M-services inside a university campus. Services offered include university events calendar, directions inside the campus, library reminders, and campus restaurants' bookings. The system is capable of detecting users' locations

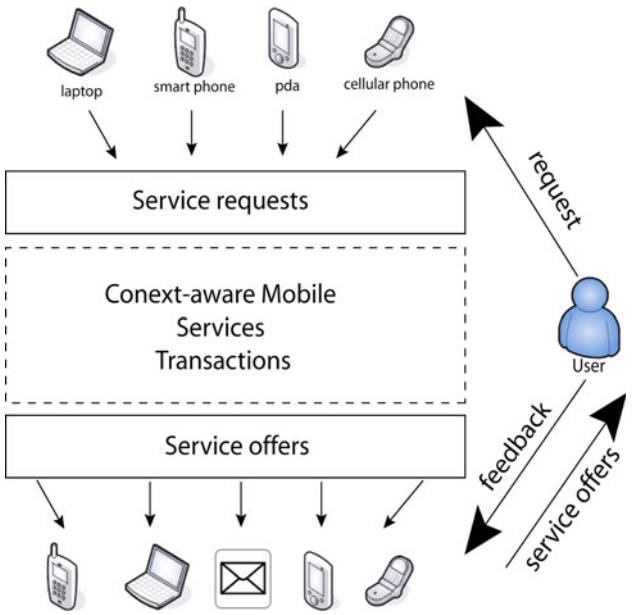

Fig. 7 Example of the context-aware transaction implementation

and communicating through a wireless infrastructure in order to gather context information and services requested by the users. For this purpose, mobile devices like iPhones, PDAs, and laptops are both offering and requesting services.

Two mechanisms are implemented: the *automatic* and the *request-based service* provisioning.

Automatic service provisioning: In this mechanism, user does not need to register to or explicitly request a service. Instead, services are automatically triggered by the MC once certain context conditions are met. The MC offers services to the user without explicit requests. For example, the library reminder service is triggered when the deadline is approaching, the restaurant's menu is sent to the user when it is lunch time and the user is not in class.

Request-based services: It corresponds to the response to the users' queries. For example, the user explicitly requests the room occupancy schedule service to check whether she can plan a meeting there.

The prototype system is developed as a proof of concept for checking the validity of the proposed model. However, this will be extended in order to include the simulation of the analytical model presented in the preceding section.

7 Conclusion and future research

This paper proposed a new model for context-aware transactions in mobile services. The aim was to manage transactions such that they comply with the required context of the desired services and adapt to the environmental conditions and users' needs. The paper also presented the

protocol that is used to execute the context-aware transactions in mobile services. The protocol ensures that context-aware transactions enforce the RACCD criteria, which is crucial to maintaining the correctness of M-services and the consistency of their underlying data. In order to test the validity of the proposed model, a prototype system has been developed. A number of performance management experiments have been conducted in order to evaluate the performance of context-aware transactions in terms of processing time, message delay, and transaction throughput.

The observation made from the evaluation is that the proposed protocol could be optimized in order to reduce the processing time and message delay and to increase the transaction throughput. We also aim to develop models in order to include failure and recovery in the performance management of context-aware transactions.

References

1. Pilioura T, Tsalgatidou A, Hadjiefthimiades S (2003) Scenarios of using web services in M-commerce. ACM SIGecom Exch 3(4):28–36
2. Benatallah B, Dumas M, Sheng QZ (2005) Facilitating the rapid development and scalable orchestration of composite web services. Distrib Parallel Databases 17(1):5–37
3. Maamar Z, Sheng QZ, Benatallah B (2003) Selection of web services for composition using location of provider hosts criterion. In: Proceedings of UMICS Workshop, Austria
4. Maamar Z, Sheng QZ, Benatallah B (2004) On composite web services provisioning in an environment of fixed and mobile computing resources. Inf Technol Manag 5(3):251–270
5. Kuramitsu K, Sakamura K (2001) Towards ubiquitous database in mobile commerce. In: Proceedings of 2nd ACM international workshop on data engineering for wireless and mobile access, California, USA
6. Younas M, Eaglestone B, Chao K-M (2004) A low latency resilient protocol for e-business transactions. Int J Web Eng Technol 1(3):278–296
7. Chen G, Kotz D (2000) A survey of context-aware mobile computing research. Dartmouth Tech. Report TR2000-381
8. Mostéfaoui SK (2005) Supporting context-aware services in pervasive environments. PhD thesis University of Fribourg
9. Malandrino D, Mazzoni F, Riboni D, Bettini C, Colajanni M, Scarano V (2010) MIMOSA: context-aware adaptation for ubiquitous web access. Pers Ubiquit Comput 14(4):301–320
10. Lin C, Jin B, Long Z, Chen H (2011) On context-aware distributed event dissemination. Pers Ubiquit Comput 5(3):305–314
11. Younas M, Eaglestone B, Holton R (2000) A review of multidatabase transactions on the web: from the ACID to the SACReD. In: Proceedings of 17th BNCOD conference, Exeter, UK, pp 140–152
12. Mostéfaoui SK, Younas M (2007) Context-oriented and transaction-based service provisioning. Int J Web Grid Serv 3(2):194–218
13. Kouadri SK, Maamar Z, Narendra NC (2006) Mobile middleware for context-aware service composition. In: Mobile Middleware, Boca Raton
14. Dey A, Abowd G, Salber D (2001) Conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum Comput Interact 16(2):97–166
15. Schilit B, Adams N, Want R (1994) Context-aware computing applications. In: Proceedings of the first workshop on mobile computing systems and applications (WMCSA), Santa Cruz, California, USA, 1994, pp 85–90
16. Younas M, Chao K-M, Anane R (2003) M-commerce transaction management with multi-agent support. In: Proceedings of the 17th international conference on advanced information networking and applications (AINA), Xi'an, China
17. Holanda M, Brayner A, Fialho S (2008) Introducing self-adaptability into transaction processing. In: Proceedings of ACM SAC'08, Fortaleza, Ceará, Brazil, pp 992–997
18. Rouvoy R, Serrano-Alvarado P, Merle P (2006) Towards context-aware transaction services. In: Proceedings of the 6th IFIP DAIS Conference, Bologna, Italy
19. Kumar V, Prabhu N, Dunham M, Seydim YA (2002) TCOT: a timeout-based mobile transaction commitment protocol. IEEE Trans Comput 5(1):1212–1218
20. Lee M, Helal S (2002) HiCoMo: high commit mobile transactions. Distrib Parallel Databases 11(1):73–92
21. Dunham MH, Helal A, Balakrishnan S (1997) A mobile transaction model that captures both the data and movement behavior. Mob Netw Appli 2:149–162
22. Tang F, Guo M, Li M, You I (2008) An adaptive context-aware transaction model for mobile and ubiquitous computing. Comput Inform 27:785–798
23. Web Services Transaction (WS-Transaction) (2005) Available online http://www-106.ibm.com/developerworks/library/ws-transpec/
24. Roberts J, Collier T, Malu P, Srinivasan K (2001) Tentative hold protocol part 2: Technical specification. Available online http://www.w3.org/TR/2001/NOTE-tenthold-2-20011128/
25. Younas M, Awan I, Chao K-M (2004) Network-centric strategy for mobile transactions. Int J Interconnect Netw 5(3):329–350
26. Younas M, Mostefaoui S.K (2010) Context-aware mobile services transactions. In: Proceedings of the 24th IEEE international conference on advanced information networking and applications (AINA), Perth, Australia, pp 705–712