A Generic Architecture Supporting Context-Aware Data and Transaction Management for Mobile Applications

Steffen Vaupel
Philipps-Universität Marburg
Hans Meerwein Straße 1
35032 Marburg, Germany
svaupel@mathematik.uni-marburg.de

Damian Wlochowitz
Philipps-Universität Marburg
Hans Meerwein Straße 1
35032 Marburg, Germany
wolochow@students.uni-marburg.de

Gabriele Taentzer
Philipps-Universität Marburg
Hans Meerwein Straße 1
35032 Marburg, Germany
taentzer@mathematik.uni-marburg.de

ABSTRACT

Mobile applications claim to operate reliably during spatial movement, however, developers have to deal with the effects of changing environmental contexts. One of the most important contexts is the connectivity of mobile devices. Since mobile applications are increasingly used as front-ends of transaction systems, they have to be designed for being able to deal with intentional or accidental loss of connection. In fact, we find a lot of mobile applications being not more than portable because they cannot operate without connections.

In order to support higher mobility – in the sense that operations may execute across the boundaries of changing network states – we discuss the problem and requirements for context-aware architectures of mobile applications. We present a generic architecture supporting users to effectively use applications on-line as well as off-line. This approach enables the concurrent execution of off-line transactions as well as their durability after synchronization. Starting from example applications, we analyze the design of existing context-aware architectures and corresponding mobile transaction models and present our approach to a generic architecture. Furthermore, we frame various conditions for advantageously using mobile transaction models.

CCS Concepts

•**Information systems** → *Distributed database transactions; Mobile information processing systems;* •**Software and its engineering** → *Software architectures;*

Keywords

Mobile data management, Mobile applications, Replication, Synchronization, Mobile transaction models

1. INTRODUCTION

Mobility and reliability during spatial movement are crucial factors for business and industry nowadays. Therefore, the question arises how architectures can support this requirement [53] [27]. App developers can choose from a variety of architectures (e.g., native, hybrid, web-based, etc.) to develop mobile apps. Sometimes, they are unaware of the impact a chosen architecture has on the mobility of an app. Web-based architectures require permanent connection to the server and therefore, are not suited to realize apps operating in an off-line mode. In contrast, native and interpretive technologies enable architectures that may be used for working temporarily without a network connection.

Changing network conditions and transaction-oriented apps can cause problems wrt. data and transactions which have to be managed in an appropriate way: (1) An app being the front-end of a multi-user transaction system has to replicate and synchronize data in order to process transactions off-line. (2) When being disconnected, the coordination of concurrent transactions is disrupted since the app is unaware of transactions performed by other users in the network. Conflicts may arise when modified replicated data is synchronized with the back-end which implies that off-line transactions can be finished at synchronization time at earliest.

The main contribution of this work is a generic architecture which uses various existing mobile transactions models to provide context-aware data and transaction management. This generic architecture uses either an off-line capable mobile transaction model in a disconnected state or an on-line transaction model in an on-line state. It supports operations like replication and synchronization being part of context-aware data management. Context-aware transaction management is carried out by a local transaction manager that may use different mobile transaction models. Since all these transaction models need replication and synchronization, we design a general replication and synchronization mechanism to support various mobile transaction models in a single architecture at the same time. As a result, developers can apply an appropriate mobile transaction model in the generic architecture to meet the requirements of context-aware apps. Moreover, we lift the context of used mobile transaction models from relational to object-oriented structures.

The presented generic architecture is evaluated wrt. varying connectivity conditions. Within a simulation, we identify connectivity conditions where mobile transaction models are preferable to standard transaction models that presume permanent connection.

This paper is structured as follows: Two example apps are presented in Section 2; they are used as running examples. We discuss the state of the art of context-aware architectures

MobileSoft'16, May 16-17 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4178-3/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2897073.2897091

and mobile transaction models in Section 3. In Section 4, we present our main contribution being a generic architecture that allows more powerful off-line usage. Section 5 contains a summary of our prototype implementation. In Section 6, the presented architecture is evaluated regarding to various conditions. In Section 7, we sum up the results and point to limitations and future work.

2. APPLICATION DOMAINS AND EXAMPLE APPLICATIONS

To illustrate our work, we present example domains where mobile applications with a context-aware architecture can be profitably used, followed by two application scenarios.

The three biggest retailers [59] in Germany in 2014, Amazon [6], Otto [14] and Zalando [17] provide mobile applications for different platforms. None of them supports transactions such as viewing and ordering products and mobile payment, in a disconnected mode. Using a replicated digital product catalog, users may view products while they are off-line. They could also order off-line and send the order later to the back-end when the connection is re-established. An inherent problem within this setting is the limitation of mobile transaction methods for off-line payments [58]. Since an order often requires payment in advance, secure payment is a crucial component of most e-commerce applications (e.g. 80% of the popular on-line shops provide PayPal [15] as a payment service [59]). Payment transactions usually contain an on-line clearing not allowing these transactions to be performed off-line. Thus, mobile e-commerce applications for off-line usage also require off-line payment methods.

In industrial settings, the problems are slightly different: Mobile apps often support or substitute manual tasks (as, e.g., inventory, order picking, and maintenance logging). The involved data objects are more individual and tailored to the surrounding business processes or real-world objects. Likewise, operations on these data objects are often more complex than in e-commerce and payment scenarios.

To sump up, context-aware architectures, and in particular, mobile transaction models are required whenever mobile apps modify either aggregated values (i.e. account balance, warehouse stock, etc.) with a set of repeatable operations (e.g., increment, decrement, etc.) or more individual objects (as they occur in e.g. a car rental) with a set of complex operations (e.g., pick-up, refuel, event of damage, etc.). An object containing only an aggregated value attribute is named *summable object*. If the set of operations contains at least one altering operation there is a conflict potential, i.e., the data objects are accessed in a competitive manner.

In the following, we will take a closer look at two specific applications – a payment app (covering an example for an aggregated value with repeatable operations) and a course booking app for fitness studio members (dealing with individual objects and more complex operations) – to demonstrate the different ways of data access. We start the discussion of each app with the assumption that there is no replication for off-line usage. This correspondents to an architecture that is not context-aware. Later in Section 4, we show implications of data replication.

2.1 Payment app

First, we consider applications for making mobile payments, like Apple Pay [7], Google Wallet [10], or PayPal [15]. Traditionally, a banking account is administered on a server at the banking site. The dataset is mainly a single value (aggregate), the account balance. This singleton is a so-called hot spot because every transaction changes or accesses this value. The set of transactions on a banking account is very limited (i.e., cash withdraw, cash deposit, debit, credit and get account balance). For each banking account, the number of users is also quite limited. Users of a banking account are the bank itself and at least one bank client. The bank itself arranges debit and credit payments from or to internal or external accounts. Banking clients may perform cash withdrawals or deposits. We assume that the account is a credit account, i.e. that it is never in the debit state (which would give raise to a conflict situation).

Example (Money transfer). We consider the following use case: A banking client wants to transfer money to another banking client from her/his mobile phone using near field communication (NFC). Most payment applications support this use case if both banking clients are on-line. The transaction is carried out as follows: The creditor delivers his/her account information to the debtor via NFC. The debtor sends a corresponding payment order to the back-end server as on-line transaction. The bank checks the cover of the payment order and executes the transaction. Finally, the creditor updates the account balance (again as on-line transaction) and confirms the transaction. If one mobile client is off-line, the payment cannot be conducted.

2.2 Course booking app

As second example, we consider a course booking app for fitness studio members. Examples of such applications are GymSync [12], GymJam [11] and BookFit [8]. Registered fitness studio members can select course spots to practice particular exercises (e.g., Yoga, Pilates, etc.) within certain time slots. The set of operations is very limited (e.g., making or canceling a reservation or checking if a course spot is available). Conflicts are very likely because each member may select every course spot. Studio members may set course preferences to indicate which courses they will select in the future. We assume that the course booking app never allows the overbooking of a course spot (conflict situation).

Example (Reservation of a course). If a fitness studio member is on-line and a selected course spot is available, a reservation transaction can be processed. If the mobile client is off-line, no transaction can be conducted.

3. CONTEXT-AWARE ARCHITECTURE DESIGN

Context-awareness of architectures is a very generally used term to describe that an architecture is aware of the context of its use [55]. We discuss the existing work, however, with the focus on transaction and data management in different connection states. In the following, we recall recommendations to the design of a connection-aware architecture and its working model including the required components. These recommendations form a kind of reference architecture. Several conceptual implementations of this reference architecture use individual mobile transaction models. Although there are plenty of different mobile transaction models, the number of available products is still limited [22]. We discuss the applicability of these products in accordance with the requirements stated above and identify shortcomings.

3.1 Connection-aware architecture design

The loss of connection caused by the *terminal mobility* [48] is not unusual [31] and should be handled by the architecture of mobile apps. In case of intentional or accidental loss of connection, it is necessary to delegate the server-located functions (called back-end) to the mobile device (called front-end) to be able to work in an off-line mode. This is sometimes called *blurring the roles*. Satyanarayanan [54] describes the resulting architecture as an *extended client server model* where the client takes over the role of the unreachable server. Pitoura and Samaras propose a similar architecture [52]. Book et al. [25] has stated that the mobility of apps is influenced by their architecture. In reverse, this means that apps have to follow a particular architecture to be connection-aware and thus off-line capable.

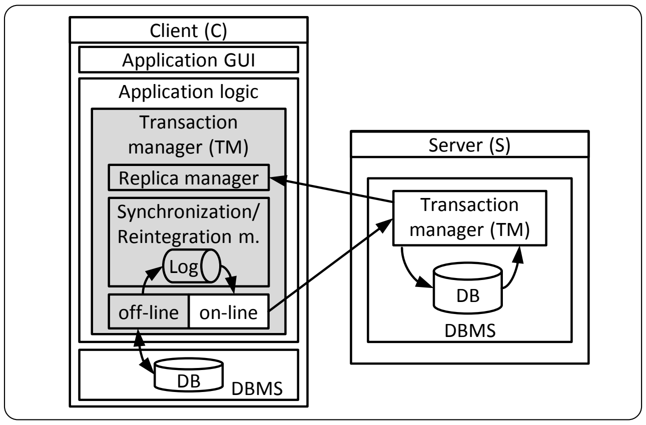

Figure 1: Extended client server model

Figure 1 shows the *extended client server model* which we use later as a blueprint of our generic architecture. The *application logic* of the client is extended by a local *transaction manager (TM)* which delegates all transactions either to the local database management system (DBMS) or to the central one on the server, dependent on its connection state. Furthermore, the *transaction manager* comprises a *replica manager*. The *replica manager* is responsible for the replication of data to be used while mobile clients are off-line. Off-line processed transactions are logged by the *synchronization/reintegration manager*. Later, this *log* is used to reprocess the transactions performed off-line.

A *transaction manager* implements a particular mobile transaction model by specifying the behavior of replication, off-line operations, and synchronization. These components (shown as hatched areas in Figure 1) are individually tailored according to the properties of the used mobile transaction model. (For example, they are asked to be conflict-free.) On-line transactions are just passed to the server and not handled by the local transaction manager. In Section 3.1.4, we will give an overview about the different mobile transaction models that have been proposed in the literature. Nevertheless, every proposed mobile transaction model results in an individual implementation of the *extended client server model*. To support a more flexible exchange of transaction models, we are heading towards a generic architecture.

3.1.1 Working model of the local trans. manager

In Figure 2, a working model is shown which distinguishes the different context states and operation steps of the local transaction manager. A mobile client starts in an online context (*On-line transaction processing*) after an optional *initial setup* of the system. In this mode, an online transaction model (often named standard transaction model) is used. Mobile clients replicate the data while they are on-line (*Replication*). The client may stay on-line after the replication or may go off-line. If so, it operates off-line (*Off-line transaction processing*) using a particular *mobile transaction model*. When the mobile client is back on-line, it must publish the modified copies (*Synchronization/Reintegration*). Hatched areas in Figure 2 denotes steps that are specific to the used mobile transaction model.

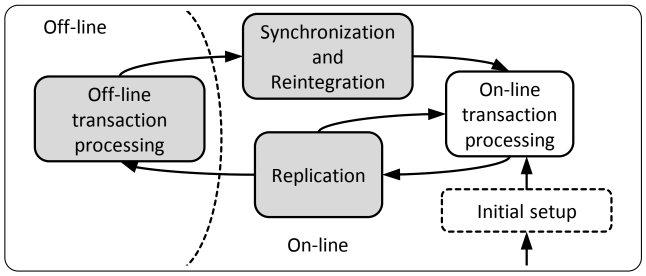
Off-line
Synchronization and Reintegration
Off-line transaction processing
On-line transaction processing
Replication
Initial setup
On-line

Figure 2: Working model of a local trans. manager

3.1.2 Anomalies

While working off-line, concurrent isolated accesses to the clients' replicas may lead to the following kinds of conflicts when synchronizing the modified data. These conflicts are called anomalies [34]:

Deletion anomaly: If a mobile client deletes a replicated record while working off-line and another client reads or changes the primary copy of this record meanwhile, a deletion anomaly occurs.

Insertion anomaly: An insertion anomaly occurs when a new record is inserted into the server DBMS and an identical record does already exist. This anomaly can occur if two mobile clients independently create a record and at least one mobile client is off-line.

Modification anomaly: The modification anomaly is the most common one. Every time when a mobile client changes a replicated record while working off-line, another client may also change it. Such a conflict may be solved by synchronizing one replica and discarding the other one. The question arises which modification is prior over the other.

Trivially, concurrent isolated read-only access of all mobile clients cannot lead to anomalies. Thus, if mobile apps use static data (e.g., encyclopedias, dictionaries, etc.) in a unidirectional way, mobility can be easily guaranteed by just replicating the used data. However, the amount of data to be replicated may be a limiting factor.

3.1.3 Replication and synchronization

Within the working model, a step for replication is needed to guarantee availability of data when being disconnected [33]. Moreover, we are heading towards applications that may change and reintegrate the replicated data into the back-end. Thus, the working model requires a step for synchronizing and reintegrating modified data after working off-line as well. Both detailed mechanisms of replication and synchronization heavily depend on the mobile transaction model used. In Section 4, we will present a generic replication strategy which is suitable for several mobile transaction models. In any way, replication strategies can be classified into *eager* and *lazy* [35] ones. Eager replication strategies try to update all copies in a single step to complete the transaction. This is inappropriate within our application area of mobile applications since mobile clients cannot be updated when being disconnected. Lazy replication asynchronously propagates replica updates to other clients after off-line transaction commitments. For doing so, the transaction is executed locally and then reprocessed later on a primary copy and other replicas. This is called *transaction-based* synchronization. Sometimes, it is sufficient to replace the primary copy with the changed replica. This kind of synchronization is called *image-based* synchronization. The synchronization approaches of concurrent accesses can be classified – similar to traditional database management systems [43] – into *pessimistic* and *optimistic* approaches. Pessimistic approaches include strategies for *conflict avoidance* while optimistic approaches provide at least *conflict detection* and often also *conflict-solving* strategies.

3.1.4 Mobile transaction models

In the following, we recall mobile transaction models that support lazy replication and pessimistic synchronization since they prevent the occurrence of anomalies and fit to the requirements outlined in the introduction very well.

In order to find mobile transaction models with these characteristics, we recall the following work: Hirsch et al. [36] survey several mobile transaction models and compare them on the basis of typical requirements for this application domain ([60], [30], [40]). Serrano et al. [57] [56] and Panda et al. [47] analyze the existing approaches in accordance to the well-known ACID (Atomicity, Consistency, Isolation, Durability) paradigm in a similar way. Mutschler and Specht [44] divide the mobile transaction models either into *first-class transaction models* (which processes transactions off-line but need to be on-line to commit the transaction) or *second-class transaction models* (which processes transactions off-line).

Based on these reviews, we discard all approaches that are not able to prevent conflicts and to work off-line like the *Kangaroo transaction model* [30], the *Preserialization Transaction Management Technique (PSTMT)* [29], the *Prewrite Transaction model* [42], the *Two-tier transaction model* [35], the *Clustering transaction model* [50] [51] [49], the *Reporting and co-transactional model* [26], and the *Isolation-only transaction model* [41].

The remaining conflict-free transaction models can be subsumed under *semantical* approaches. Semantical approaches use the structure of the data or semantical properties of transactions performed on replicas [32]. We have selected the *Keypool transaction model* and the *Escrow transaction model* for use in our generic architecture.

The number of products is still limited, outdated, and very homogeneous in terms of used mobile transaction models. OracleLite [5], IBM DB2 Everyplace [1], Microsoft SQL Server CE [13], and Sybase Adaptive Server Anywhere [3] are some commercial mobile database systems (mDBMS). Their architecture correspond in general to the *extended client server model.* All products use an *image-based* synchronization and do not support conflict prevention. Thus, durable off-line transactions cannot be carried out.

With regard to the available products, we assume that the replication is possible within our scenarios but does not yet include a conflict-avoiding mechanism while processing transactions off-line. This behavior is illustrated at the following examples.

Example (Payment app): A debit transaction decreases the replicated *account* value of the debtor and increases the replicated *account* value of the creditor. The app checks the coverage of the replicated *account* value locally. The transaction can happen off-line via NFC. At a later date, the banking client reprocesses the debit or credit transaction on the primary copy in order to synchronize the account balance (on-line transaction). However, if the debtor withdraws money and changes the primary copy before executing the synchronization, the coverage of the account cannot be ensured. The bank is unaware that the customer has already transfered money from the replicated *account.* Since the account may be in debit state, a conflict may arise.

Example (Course booking app): The fitness studio member uses a copy of the entire data set, i.e., of all course spots. A reservation transaction checks whether a course spot is unselected by other members and selects it. At a later date, the fitness studio member(s) synchronize the changed course spots with the primary copies. If another fitness studio member selected the same course spot, the transaction of one member gets lost during synchronization. Nevertheless both members get a local commit of their transaction. Since a course spot may be overbooked, a conflict may arise.

3.2 Problem statement

Although we have conflict-preventing mobile transaction models, the available products do not use them. As stated by Gollmick [34], barriers are the demarcation of the mobile database management systems (mDBMS) and the semantics of transactions located at the mobile client or at the server. Either the mobile application realizes a mobile transaction concept on the level of application logic, or the mobile database management system supports a seamless interface to use the semantical information of transactions being defined by application logic. With the focus on mobile development, the following question arises: How does a generic architecture for connection-aware mobile applications looks like that allows various mobile transaction models (RQ1)?

The existing work of mobile transactions models focuses on relational data models. Following object-oriented design, data models of mobile applications are object-oriented ones (i.e. class models). Thus, the existing concepts must be rethought and adapted to the context of object-oriented data modeling. App developers are often familiar with the object-relational mapping (ORM) to serialize objects into relations but unsettled in applying this concept in the context of mobile transactions involving replication and synchronization. Therefore, the next question is: Can mobile transaction models be applied in the context of object-oriented application development and what are the effects (RQ2)?

Finally, given mobile transaction models have not been evaluated well. From the perspective of an app developer, the conditions (e.g., connectivity, number of users and data, etc.) under which mobile transaction models should be used are unclear. Mobile transaction models may bring profit to disconnected clients but may also cause additional costs (wrt. replication and synchronization). They may cause reduced performance for highly connected users (clients). These considerations lead us to the third question: Which kinds of context conditions are assumed for an app to profit from using mobile transaction models (RQ3)?

4. A GENERIC ARCHITECTURE FOR CONNECTION-AWARE APPS

Based on the *extended client server model* and the working model presented in Section 3, we present a generic architecture for connection-aware apps that can be instantiated with various transaction models. In this section, we focus on the instantiation with conflict-free models, namely Keypool and Escrow. They seem to be especially promising for context-aware transaction processing. Knowing the differences between these mobile transaction models in terms of their individual replication and synchronization, we can modify the working model in order to use both mobile transaction models in a single generic architecture. Finally, we present the developed design along the steps of a modified working model.

4.1 Conflict-free mobile transaction models

We focus on conflict-free mobile transaction models according to the requirements of our example apps. The selected models use different strategies to prevent conflicts:

4.1.1 Keypool transaction model

The *Keypool* transaction model [2, 4] uses the structure of the given dataset. The basic idea of the Keypool method is to split the entire dataset into subsets that are distributed among the participating mobile clients. Figure 3 illustrates a data split to three mobile clients within the replication step. Every client gets an amount of data that is exclusively replicated. When a client is off-line, he/she can operate on the replicated data without limitations. Within the synchronization step, the partial data is reintegrated into the primary copy using *image-based* synchronization. Independent of the operation to be performed while being off-line, the result can be adopted by substituting the value of the primary copy for the value of the changed replica (i.e, the *image*). The Keypool approach avoids *deletion* and *modification* anomalies by design. Without additional provision, *insertion* anomalies may occur. During the course of this work, we ensured that insertion anomalies cannot occur by the use of an object-relational mapping framework.

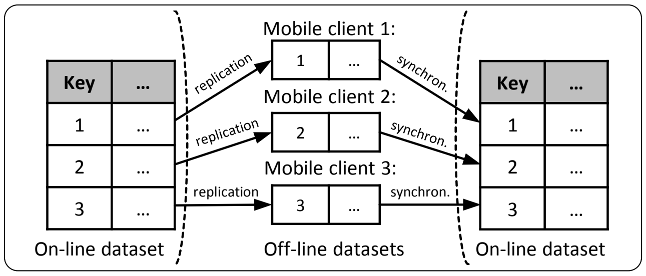

Figure 3: Keypool replication and synchronization

4.1.2 Escrow transaction model

The *Escrow* transaction model [45, 39] is well suited to access and modify aggregate data. The basic idea of the Escrow method is to restrict the set of transactions and/or the domains of their arguments when being performed off-line. While the Keypool transaction model splits the dataset and thereby, may risk to provide an undersized or empty dataset, the Escrow approach always provides the full dataset. Figure 4 shows the replication scheme of the data to two mobile clients. Every client gets a full copy of the dataset. Assuming that the semantic of provided transactions is known, every record is transformed at the step of replication such that conflicts cannot occur at the synchronization step. Considering mobile payment, for example, the debit transaction may cause conflicts. Therefore, the domain of its argument is restricted such that just small amount may be withdrawn. One possible strategy is to equally distribute the amount among all participating clients as shown for the example aggregate values in Figure 4. Since several mobile clients may change the same value, this strategy always guarantees conflict-free synchronization afterwards.

An *image-based* integration does not work here since either one or another image can be written back to the primary copy but not both. The other values would be lost (called Lost-update [21, 19]). Thus, the reintegration of changed values has to be based on a *transaction-based* approach. It collects all transactions performed off-line and replays them on the primary copy. The repeated transactions must have the same effects as being performed on-line but usually do not achieve the same value on the primary copy as on the replicated copies. This property is called *semantical serializability* [46]. To ensure it, all operations must be repeatable (such as decrement and increment) and their semantics on restricted values has to correspond to the one on non-restricted ones. The Escrow approach avoids *insertion*, *deletion*, and *modification* anomalies by design. A generalization beyond aggregated values is the PRO-MOTION transaction model [61]. Within that approach, so-called *compacts* form local constraints and guarantee semantical serializability.

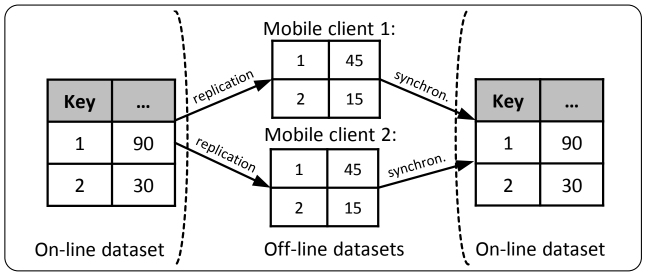

Figure 4: Escrow replication and synchronization

4.2 Modification of the working model

As presented in Section 3.1.1, replication and synchronization are the major steps of the working model. The mobile transaction models Keypool and Escrow use their own approaches to replicate and synchronize data. Keypool replicates data sets by splitting, while keeping their values unchanged; Escrow, however, does not split the dataset but replicates it by changing every value of the dataset in accordance with the number of mobile clients. Synchronization is performed *image-based* by the Keypool method and *transaction-based* by the Escrow method. Hence, the replication and synchronization steps of these mobile transaction models cannot be mixed for providing more than one mobile transaction model in a single architecture. Each mobile transaction model needs an individual implementation.

In order to circumvent this problem, we modify the working model by adding the following conditions: (1) The replication step is not allowed to limit the dataset or to transform its values. If a mobile transaction model requires a limited or modified set of data, the dataset must be preprocessed accordingly within the off-line transaction processing step. (2) If a synchronization method is more powerful than another one (i.e., *transaction-based* covers *image-based* synchronization), the weaker method can be substituted by the stronger one. A set of mobile transaction models that satisfies these two conditions can be applied in our architecture, like the Keypool and Escrow method. Furthermore, most of the conflicting mobile transaction models satisfy these conditions.

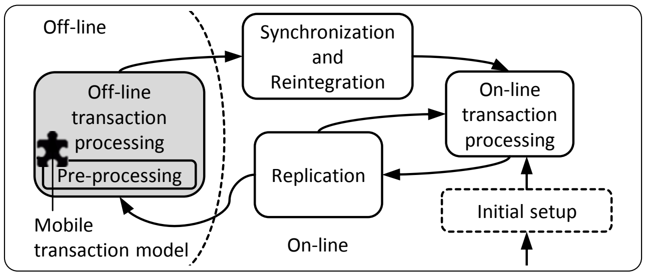
Off-line
Synchronization and Reintegration
Off-line transaction processing
Pre-processing
On-line transaction processing
Replication
Initial setup
Mobile transaction model
On-line

Figure 5: Modified working model

Figure 5 shows the working model of local transaction managers being used in our architecture. *Replication*, *Synchronization and Reintegration* are independent generic steps, while the *off-line transaction processing* implements the mobile transaction model including some pre-processing (shaded area). Different mobile transaction models like Keypool and Escrow, may be plugged in and work independently to the steps performed on-line.

4.2.1 Data modeling and initial setup

Mobile applications are usually designed in an object-oriented way. Hence, the data model does not define relations but object classes. If a database is used underneath, a class model can be translated into a relational data model by object-relational mappers (ORM). ORM frameworks can create empty database schemes from class models, and convert objects into table rows. Vice versa, database records may be translated back to object structures. In our presented data models, classes and attributes may be annotated by an asterisk to indicate that objects of a certain class should be split (Keypool) or that an attribute is an aggregate (Escrow). (See Figures 6 and 7 below.)

According to Figure 1, the server-sided architecture is lightweight (i.e., no server facilities to set up the database), but it needs at least a database management system or a similar service to persist data. In our prototype implementation (described below), we use a relational database MySQL 5.6 at the server side. The initial setup is triggered and performed remotely by a mobile client. It can only be performed once by the first appearing mobile client.

Example (Payment app): Figure 6 shows the data model of the payment app. It consists of the class *Account* only. A valid instance of this data model may have only one *Account* object; the asterisk at the attribute *amount* indicates that this object may be split and allocated to mobile clients. Thus, mobile clients may share this account object, particularly the attribute *amount*.

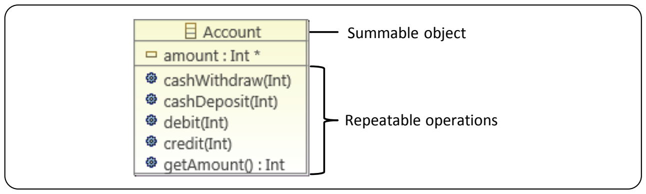

Figure 6: Data model of the payment app

Example (Course booking app): Figure 7 shows the data model of the course booking app. It contains course spots and persons. Course spots are not summable since each course spot is an individual object. This kind of modeling allows using the Keypool approach. However, it may lead to a large set of objects being difficult to handle. A reservation of a course spot is made by setting the participant pointer to a person. The asterisk at class *CourseSpot* denotes that its objects may be distributed among mobile clients.

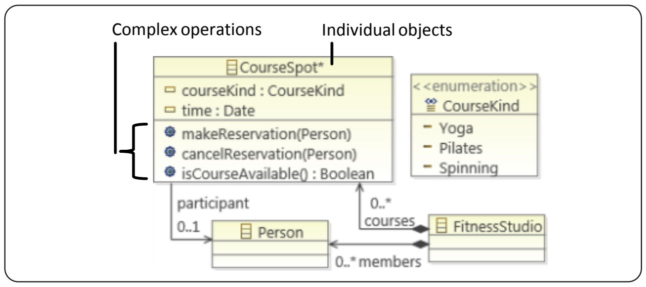

Figure 7: Data model of the course booking app

4.2.2 On-line Transaction Processing

The on-line transaction processing step of our generic architecture is not affected by the changes made to the working model. It operates as stated in Section 3.1.1.

4.2.3 Replication

Due to working model modifications, the replication step of our generic architecture copies the entire set of data to the mobile clients (*full replication*). The replication step is – similar to the initial setup of the server-located database – triggered by the clients. This is called *pull-based replication* in opposite to the *push-based replication* [37].

4.2.4 Off-line Transaction Processing

The off-line transaction processing is the crucial part of our generic architecture. Replication is required to operate off-line but also involves the risk of synchronization conflicts. The selected mobile transaction models should ensure conflict-free synchronization and commit transactions immediate without having to wait for reconnection.

Synchronization conflicts may occur every time when a schedule (i.e. a sequence of off-line transactions on replicas) cannot be *serialized* on the primary copy. Such a schedule is *serializable* if it is equal to a serial schedule on the primary copy. The most-commonly used definitions of this equivalence concern the order of reading and writing operations (conflict equivalence) or the view relation (view equivalence) on relational variables. This is hardly applicable to replicated isolated objects because mobile clients cannot communicate while they execute the transaction and therefore they cannot detect conflicts with other isolated mobile clients [24]. The mobile clients only know how many concurrent mobile clients are registered and which operations they may perform. Based on that fact, a conflict matrix can be set up in advance. Tables 1 and 2 show which conflicts may occur between the transactions involved in our example apps. We consider only transactions finished successfully.

Table 1: Conflicts in the payment app[1]

t_1/t_2	`debit`	`credit`	`getAmount`
`debit`	Yes	No	Yes (No)[2]
`credit`	No	No	Yes (No)[2]
`getAmount`	Yes (No)[2]	Yes (No)[2]	No

Table 2: Conflicts in the course booking app

t_1/t_2	`makeRes.`	`cancelRes.`	`isCourseAv.`
`makeRes.`	Yes	Yes	Yes (No)[2]
`cancelRes.`	Yes	Yes	Yes (No)[2]
`isCourseAv.`	Yes (No)[2]	Yes (No)[2]	No

A conflict is given, if at least two transactions t_1 and t_2 are performed *successfully* off-line and cannot be re-processed later on the primary copy in an arbitrarily chosen order (no state commutativity) or the transactions return values dependent on the re-processing order (no return value commutativity [62]).

Requiring state commutativity and return value commutativity implies strict read and write operations. Dependent on the app, it might be appropriate to relax the conflict definition and tolerate weak read but strict write operations [50]. For the course booking app, for example, the operation *isCourseAvailable* may use inconsistent copies, but the operation *makeReservation* is allowed only on consistent copies.

Using the Escrow method, the *aggregate* that is processed by conflicting off-line transactions is divided among the set of mobile clients beforehand. Besides a full allocation of the *aggregate* to the mobile clients, it can also be limited by a factor so that a part of the aggregate may remain unallocated. In case of the payment app, the amount is distributed among the set of mobile clients. All clients can access the summable *account* object. But the conflicting off-line transactions accept just a limited domain of argument values (e.g. a limited debit amount) compared to the on-line processing.

The replication strategy for the Keypool method filters the whole set of objects. The filter function maps objects uniquely to mobile clients but can also keep objects unallocated. *Round-Robin* is a example filter function that assigns data objects to one after the other mobile client and starts again at the first client as long as objects are available. To allocate preferred objects, mobile clients may indicate their intention with so-called *preference sets*. In that case, the filter function maps objects according to the users' preferences if possible. However, filtered-out objects cannot be accessed in a disconnected state, and consequently, cannot be involved in synchronization conflicts. For the course booking app, every course spot is mapped to one mobile client at most. Every mobile client can work on a disjoint subset of objects. All the transactions processed off-line are logged for both mobile transaction models to allow the subsequent synchronization after re-connection (as explained below).

[1]The transactions cash debit and cash credit are discharged for the mobile client.

[2]Relaxed conflict definition (tolerate weak read).

Example (Money transfer): In this scenario a couple shares a banking account. Before accessing the account object off-line, the amount value (aggregate) will be divided into equal amounts. For example, an amount of 100$ will be divided into 50$ located at one person (client 1) and 50$ at the other person (client 2). Both can transfer 50$ off-line. In doing so, the account is always on the credit side. The downside of this strategy is that they cannot transfer 100$ (on-line or off-line) although if they have the amount of money.

Example (Reserving a course spot): Given a set of course spots being replicated to a set of mobile clients, the filtering functions maps the course spots according to the preferences of fitness studio members. That way, the subset of course spots assigned to a mobile client may contain a preferred spot off-line (to make a reservation). Thus, an off-line transaction can take place. Hence, a fitness studio member can make his/her reservation off-line without causing a conflict during synchronization. A disadvantage of this strategy is that a fitness studio member cannot reserve a course spot which is not mapped to his/her identification, even if the spot is not selected by any other member. The simulation will show how often these reservations remains unused.

4.2.5 Synchronization and reintegration

Modified replica must be synchronized with the primary copy and the replicas of the other mobile clients finally.

Synchronization scheme: The proposed generic architecture use a hub-oriented synchronization scheme. Every mobile client synchronizes modified replicas with the primary copy on the server (hub). Since the server does not push the announced changes to the mobile clients, the mobile clients are asked to pull all the changes from the server.

Image-based synchronization: As mentioned before, all commercial products use an image-based synchronization which changes the primary copy with the image (or value) of the modified replica. The semantics of the performed transaction is not necessary to know for this synchronization method. Unfortunately, the image-based synchronization requires an $(n-1)$ consistency of the replicas which requires that at most one replica is modified. A consistency less than $(n-1)$ produces more than one modified image which cannot all be synchronized with the primary copy (i.e., lost updates would occur). Image-based synchronization is sufficient for the Keypool method because an object is accessed by at most one mobile client.

Transaction-based synchronization: A rarely used synchronization method is the transaction-based method which synchronizes the primary copy by reprocessing all the transactions performed on the replicas. This is possible if every transaction performed off-line preserves the precondition of every other transaction performed off-line. Thus the synchronization method requires that the semantical effects of all the performed transactions are known. The transaction-based synchronization does not require any consistency level and may integrate fully inconsistent sets of replicas. Transaction-based synchronization is adequate for the Escrow method because the distribution of the aggregate preserves the preconditions of the involved conflicting transactions. Since the transaction-based synchronization method covers the image-based synchronization method, the transaction-based synchronization is used for both transaction models within our architectural design assuming that the semantics of all transactions is known.

5. PROTOTYPE ARCHITECTURE

The prototype implementation provides a proof-of-concept of how a generic architecture for mobile apps can look like and how several mobile transaction models can be provided in a single architecture (RQ1). Moreover, the prototype uses object-oriented data modeling. We thus demonstrate that mobile transaction models can be applied in the context of object-oriented application development (RQ2). Overall, we carried out a working prototype for Android that realizes the generic architecture described in the previous section.

We use the Eclipse Modeling Framework (EMF) [9] as object-oriented data modeling language. EMF allows us to generate code for object-oriented data management from a domain model. The object-relational mapping framework Hibernate [20], in combination with the model-relational mapping framework Teneo [16], allows setting up a data base scheme and persisting object structures in a server-located database. These object structures can be modeled as instances of the given domain model. Full data replication can be realized by loading the model instances from the relational database (server) and storing them locally on the mobile devices in a file in XML/XMI format. Since Hibernate is a certified JPA-Provider [38] [28], it includes a transaction session management that can be reused to handle the on-line transaction processing of the mobile clients. The transaction-based synchronization is also handled as on-line transaction. According to the selected transaction model, single objects are transformed (Escrow) or multiple object sets are filtered (Keypool) in a preprocessing step of the off-line transaction processing.

6. EVALUATION

In the following, we evaluate the design of our generic architecture to show its applicability and usefulness. To check under which connectivity conditions the proposed generic architecture works best, we designed several experiments. We built up a separate simulation system and investigate correlations between key parameters such as number of clients, their activity, the connectivity, and number of objects. The results answer under which conditions the proposed generic architecture is used best (RQ3).

6.1 Simulation design

The shown examples are limited to only a few mobile users due to the limitation of parallel Android emulation instances. Thus, such a test can only demonstrate the applicability and soundness rather than the usefulness of the generic architecture. For a broader evaluation in terms of scalability and correlations between key parameters (i.e. numbers of objects and users and levels of activity and connectivity) we use an external simulation system. It behaves equally to the prototype implementation from a functional perspective. The evaluation results are available at [18].

6.1.1 Method

The simulator monitors the throughput, i.e., the number of transactions, using a standard on-line transaction architecture (as used in web-apps) on the one hand and the proposed generic architecture on the other hand. Table 3 shows the independent simulation variables as simulation input.

Table 3: Independent simulation variables

Name	Description	Domain
#Subjects	The number of mobile clients.	1 .. 7560
#Objects	The number of objects or aggregate size.	1 .. 3780
#AllocObjects	The number of allocated objects or elements of an aggregate.	0 .. #Objects
Activity	The activity level of a subject.	0..100
Connectivity	The connectivity level of a subject.	0..100
#Preferred	The number of objects or elements preferred by the subject.	0 .. #Objects

The number of mobile clients (#Subjects), number of objects or aggregate size (#Objects) and the number of allocated objects or allocated elements from an aggregate (#AllocObjects) are global subject-independent variables. The variables *Activity* and *Connectivity* are subject-specific, and therefore, called local variables. This means that different subjects may have different activity and connectivity levels. The *connectivity* determines the duration of on- / off-line periods in an abstract way. Basically, subjects may access objects or elements of an aggregate randomly. The access may also be limited by another subject-specific variable called #Preferred. If #Preferred:=1, a mobile client always selects the same object; random access from the entire set of objects is expressed by #Preferred:=0 and from a limited set of objects by #Preferred > 0. #Preferred determines the degree of interference among different mobile clients.

The reader may notice that the processed operations are not further specified. Simulating the Keypool method, we use abstract simulation objects with write and read operations. Conflicting transactions are considered as write operation while non-conflicting ones form as read operation. Simulating the Escrow method, we use an abstract aggregate object with increment and decrement as operations. Inside one simulation step, mobile clients access the aggregate or a set of objects according to their local settings (i.e., Activity, Connectivity, and #Preferred). They either act on their locally stored replicated objects (off-line transaction) or on the primary copy (on-line transaction).

Table 4: Dependent simulation variables

Name	Description
#On-lineProcessed	Number of transactions processed in connected mode.
#On-lineRejected (Unused)	Number of transactions rejected in connected mode because the object was locked and not used by the corresponding client in a disconnected mode.
#Off-lineProcessed	Number of transactions processed in disconnected mode.

After transaction processing, dependent variables are determined as simulation output (see Table 4).

The goodput (number of successfully finished transactions) of the standard architectures working only on-line is defined as:

$$\text{Goodput(On-line)} := \#\text{On-lineProcessed} \quad (1)$$

The goodput of our generic architecture working on-line and off-line is defined as:

$$\begin{aligned}\text{Goodput(On-/Off-line)} :=& \#\text{On-lineProcessed} + \\ & \#\text{Off-lineProcessed} - \quad (2) \\ & \#\text{On-lineRejected (Unused)}\end{aligned}$$

We can see that the number of off-line processed transactions (#Off-lineProcessed) may increase the overall goodput. By assigning data to mobile clients exclusively, however, on-line transactions may be rejected which may reduce the overall goodput. Therefore, it is very important to understand under which conditions the combined on-line/off-line processing yields an overall benefit. Our simulation experiments described below answer this question.

6.1.2 *Internal and external validity*

To ensure the internal validity of the simulation, all effects on the set of dependent variables must depend on modifications of only one independent variable in order to exclude a cause-effect conclusion [23]. Each of our simulation experiments fulfills this requirement. Moreover, we repeated our simulation 10.000 times per experiment. The results are reproducible and corrected wrt. statistical outliers.

Since we use abstract objects for the simulation, the simulation is generalizable. While simulations of the Keypool method do not introduce any restrictions wrt. objects and transactions, the Escrow model simulations cover increment and decrement operations on aggregates only. Another threat to external validity is the use of randomized activity and connectivity of the mobile clients. In order to increase external validity it would be helpful to have empirical, i.e. real world, data sets of mobile clients' connectivity and activity.

6.2 Simulation experiments and observations

In the following, we mainly report on simulations of the Escrow method since Keypool-based simulations show similar results as pointed below.

6.2.1 *Escrow*

Experiment 1 – Payment app. First, we simulate the payment app with an aggregate value of 100$ and an equal split of the aggregate (50$ at the server and 50 $ at the client). The client prefers `debit` and `credit` amounts from 1 to 100 $. Figure 8 shows the results of this experiment. The goodput of the standard architecture raises from 0% (disconnected) to 100% (fully connected). The goodput of the proposed generic architecture starts at 63.26% (disconnected) and reaches also 100% (fully connected). Hence, this experiment shows that the proposed generic architecture improves the number of processed transactions in weak connection states up to 62.26%.

Experiment 2 – Homogeneous multi-subjects. In contrast to the prior experiment, we split the aggregate now equally to 25 clients (subjects). Figure 9 shows the results for a multi-user payment app. We find similar results as for Experiment 1, however, the goodput is slightly smaller since the increased number of mobile clients causes more conflicts.

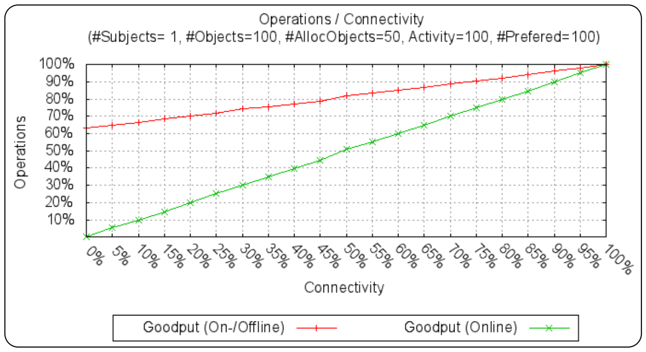

Figure 8: Goodput for the payment app

The goodput of the proposed generic architecture is up to 51.17 % higher than for the standard architecture.

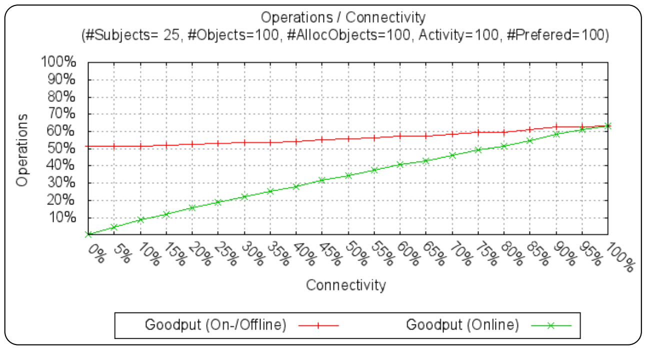

Figure 9: Goodput for a multi-user payment app

Experiment 3 – Homogeneous multi-subjects' scalability. Given the settings of Experiment 2, we check scalability and change either the amount of subjects or the amount of objects.

Table 5: Homogeneous multi-subject scalability

#Subjects	#Objects	Gain
1	100	74.85%
2	100	62.29%
50	100	50.74%
100	100	50.29%
200	100	50.26%
500	100	50.12%
100	1	66.72%
100	2	60.02%
100	50	50.44%
100	100	50.26%
100	200	50.40%
100	500	50.34%

Table 5 shows these scalability measurements; the column *gain* reports the benefit of the proposed generic architecture. It scales up with the numbers of users and objects. In a disconnected mode, the effort of the proposed generic architecture converges to 50%, since 50% of the processed transactions (i.e. all credit transactions) are conflict-free. We conjecture that the convergence point is determined by the ratio of conflicting and non-conflicting transactions.

Experiment 4 – Homogeneous multi-subjects' activity and connectivity. Given the settings of Experiment 2, we test the relation of connectivity levels with activity levels now. Figure 10 shows that the goodput of the proposed generic architecture covers the goodput of the standard architecture even if the activity level is reduced. (Note that it was 100 % in Experiment 2.) In general, the goodputs of both architectures decrease correspondingly with reduced activity.

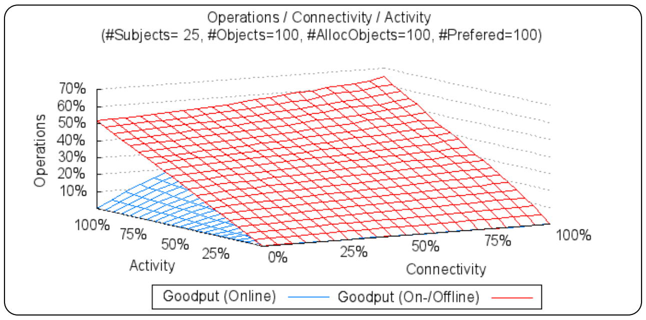

Figure 10: Decrease of goodputs according to decrease of activity

Experiment 5 – Heterogeneous multi-subjects. The prior experiments used homogeneous subjects which leads to nearly identical plots for every subject. Since mobile clients usually differ in their network connectivity, their activities and their preferences of choice, we designed an experiment with heterogeneous sets of subjects controlled by subject-specific independent variables (see Table 3).

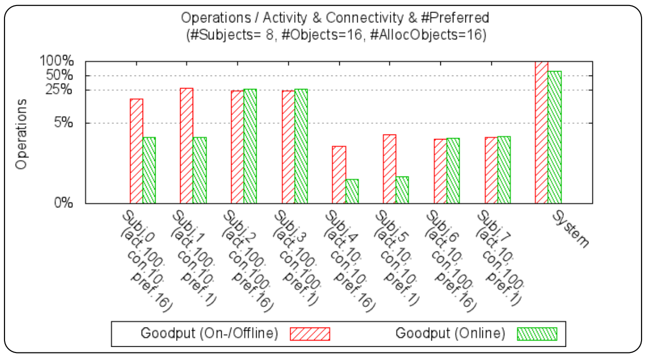

Figure 11: Individial goodputs of subjects according to their subject-specific independent variables

As shown in Figure 11, the overall goodput as the sum of the subjects goodput (system bar) of the proposed generic architecture is better than the goodput of the standard architecture. However, the results differ considerably wrt. activity and connectivity levels as well as size of the preferred set. Figure 11 shows the corresponding simulation results. Our findings are these: The proposed generic architecture is most profitable for mobile clients with high activity, poor connection and a small set of preferred objects to be accessed. The proposed generic architecture is not profitable for mobile clients with high activity and strong connection.

6.2.2 *Keypool*

For the simulation of the Keypool transaction model very similar experiments have been carried out yielding similar findings as well. Therefore, we present just one experiment: *Experiment 1 – Course booking app.* Within this simulation, we assume a fitness studio with 18 hours of opening (7 days) and 3 parallel courses with 10 participants on average yielding 3780 course spots. We assume that the fitness studio has 7560 members. Members may select three courses as favorites. The goodput of the proposed generic architecture is up to 57.61% higher than for the standard architecture (shown in Figure 12). As mentioned earlier, the generic architecture can also handle optimistic mobile transaction models. Thus, we can estimate the goodput of an optimistic mobile transaction model for the given scenario, which is much higher than the goodput of the pessimistic approach.

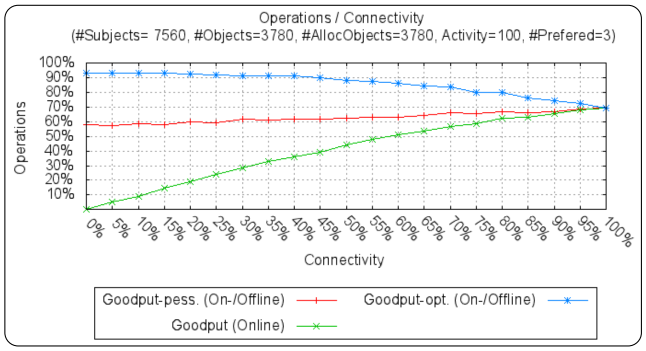

Figure 12: Goodput for the course booking app

7. RELATED WORK AND CONCLUSION

To our knowledge, there is no approach that supports conflict-free and context-aware data and transaction management in a generic architecture. The PRO-MOTION transaction model [61] is the most similar approach. This model uses user-defined constraints evaluated by the local transaction manager. However, conflicts and anomalies may arise due to insufficiently-defined constraints.

We presented a generic architecture that supports conflict-free and context-aware data and transaction management. This architecture can integrate different conflict-free mobile transactions models. Transactional anomalies arising in existing mobile database systems by accidentally using the same identities are avoided in our approach since the architectural design is object-oriented and the mobile transaction models used are adapted to this design. The applicability was shown with a prototypical implementation for Android. Furthermore, we demonstrated the usefulness of the proposed generic architecture by conducting several simulation experiments. We identified clear context conditions for the advantageous use of the proposed generic architecture. The proposed generic architecture is most profitable for mobile clients with high activity, poor connection conditions and a small set of preferred objects to be accessed. However, the maximum number of expected clients in a life-cycle must be known beforehand in order to allocate the data, i.e., it does not currently support the dynamic entry of additional clients. Furthermore, conflicts between transactions are identified *manually.* Tool support for automatic conflict detection would be helpful. In future work, our generic architecture shall be used in practical mobile software development to confirm our findings in practice.

8. REFERENCES

[1] IBM DB2 Everyplace Installation and User's Guide, October 2003.

[2] Adaptive Server ® Anywhere 9.0.2 – Adaptive Server Anywhere SQL User's Guide, October 2004.

[3] Adaptive Server ® Anywhere 9.0.2 – Introducing SQL Anywhere ® Studio, October 2004.

[4] Adaptive Server ® Anywhere 9.0.2 – SQL Remote ™ User's Guide, October 2004.

[5] Oracle ® Database Lite Getting Started Guide, February 2010.

[6] Amazon Europe Core S.à r.l., 2015. http://www.amazon.de.

[7] Apple Pay, 2015. http://www.apple.com/apple-pay.

[8] BookFit, 2015. http://www.bookfitapp.co.uk/.

[9] Eclipse Modeling Framework (EMF), December 2015. https://www.eclipse.org/modeling/emf – Version 2.12.

[10] Google Wallet – Google Payment Corp. (GPC), 2015. https://www.google.com/wallet/.

[11] GymJam, 2015. http://www.thegymjam.com/.

[12] GymSync, 2015. http://www.gymsync.co.uk/.

[13] Microsoft – Beginner's Guide to SQL Server Compact, December 2015.

[14] Otto (GmbH & Co KG), 2015. https://www.otto.de.

[15] PayPal (Europe) S.à r.l. et Cie, S.C.A., 2015. https://www.paypal.com/de.

[16] Teneo, 2015. https://wiki.eclipse.org/Teneo – Version 2.0.0.

[17] Zalando SE, 2015. https://www.zalando.de.

[18] Experimental results of mobile transaction model simulations, January 2016. http://www.uni-marburg.de/fb12/swt/forschung/mobilesoft16/evaluation/.

[19] A. Adya, B. Liskov, and P. O. Neil. Generalized isolation level definitions. In D. B. Lomet and G. Weikum, editors, *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA*, pages 67–78. Institute of Electrical and Electronics Engineers (IEEE), March 2000.

[20] R. F. Beeger, A. Haase, S. Roock, and S. Sanitz. *Hibernate - Persistenz in Java-Systemen mit Hibernate 3*. dpunkt.verlag, 2006.

[21] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In M. Carey and D. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD international conference on Management of data, San Jose, CA, USA, May 22 - 25, 1995*, volume 24, pages 1–10. Association for Computing Machinery (ACM), May 1995.

[22] G. Bernard, J. Ben-Othman, L. Bouganim, G. Canals, S. Chabridon, B. Defude, J. Ferrié, S. Gançarski, R. Guerraoui, P. Molli, et al. Mobile databases: a selection of open issues and research directions. *ACM Sigmod Record*, 33(2):78–83, June 2004.

[23] B. Bernard Nicolau de França and G. Horta Travassos. Simulation based studies in software engineering: A matter of validity. *CLEI Electronic Journal*, 18(1):5–5, April 2015.

[24] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[25] M. Book, V. Gruhn, M. Hülder, and C. Schäfer. Der Einfluss verschiedener Mobilitätsgrade auf die Architektur von Informationssystemen. In J. F. Hampe, F. Lehner, K. Pousttchi, K. Ranneberg, and K. Turowski, editors, *Mobile Business- Processes, Platforms, Payments, Proceedings zur 5. Konferenz Mobile Commerce Technologien und Anwendungen (MCTA 2005) in Augsburg*, volume 59 of *LNI*, pages 117–130. Gesellschaft für Informatik (GI), 2005.

[26] P. K. Chrysanthis. Transaction processing in mobile computing environment. In B. Bhargava, editor, *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, volume 82, pages 77 – 82. Institute of Electrical and Electronics Engineers (IEEE), October 1993.

[27] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, LNCS 7475, pages 1–32. Springer, 2013.

[28] L. DeMichiel. Java persistence 2.0 specification, November 2009.

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

[30] M. H. Dunham, A. Helal, and S. Balakrishnan. A mobile transaction model that captures both the data and movement behavior. *Mobile Networks and Applications*, 2(2):149–162, 1997.

[31] T. Fuchß. *Mobile Computing: Grundlagen und Konzepte für mobile Anwendungen*. Hanser, Carl, 2009.

[32] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems (TODS)*, 8(2):186–213, 1983.

[33] C. Gollmick. Replication in mobile database environments: A client-oriented approach. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA '03)*, page 980. Institute of Electrical and Electronics Engineers (IEEE), 2003.

[34] C. Gollmick. *Konzept, Realisierung und Anwendung nutzerdefinierter Replikation in mobilen Datenbanksystemen*. PhD thesis, Friedrich Schiller University of Jena, 2006.

[35] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In J. Widom, editor, *Proceedings of the 1996 ACM SIGMOD international conference on Management of data (SIGMOD '96)*, volume 25, pages 173–182. Association for Computing Machinery (ACM), June 1996.

[36] R. Hirsch, A. Coratella, M. Felder, and E. Rodriguez. A framework for analyzing mobile transaction models. *Journal of Database Management*, 12(3):36, 2001.

[37] Z. Itani, H. Diab, and H. Artail. Efficient pull based replication and synchronization for mobile databases. In *International Conference on Pervasive Services*

2005, pages 401–404. Institute of Electrical and Electronics Engineers (IEEE), 2005.

[38] M. Keith and M. Schincariol. *Pro JPA 2*. Books for professionals by professionals. Apress, 2013.

[39] F. Laux and T. Lessner. Escrow serializability and reconciliation in mobile computing using semantic properties. *International Journal On Advances in Telecommunications*, 2(2):72–87, 2009.

[40] J. Lee and K. Simpson. A high performance transaction processing algorithm for mobile computing. In H. Adeli, editor, *Proceedings Intelligent Information Systems. IIS'97*, pages 486–491. Institute of Electrical and Electronics Engineers (IEEE), 1997.

[41] Q. Lu and M. Satyanaranyanan. Isolation-only transactions for mobile computing. *ACM SIGOPS Operating Systems Review*, 28(2):81–87, April 1994.

[42] S. K. Madria and B. Bhargava. A transaction model to improve data availability in mobile computing. *Distributed and Parallel Databases*, 10(2):127–160, 2001.

[43] D. A. Menascé and T. Nakanishi. Optimistic versus pessimistic concurrency control mechanisms in database management systems. *Information systems*, 7(1):13–27, 1982.

[44] B. Mutschler and G. Specht. *Mobile Datenbanksysteme: Architektur, Implementierung, Konzepte*. Xpert.press. Springer Berlin Heidelberg, 2013.

[45] P. E. O'Neil. The escrow transactional method. *ACM Transactions on Database Systems (TODS)*, 11(4):405–430, December 1986.

[46] M. Ouzzani, B. Medjahed, and A. K. Elmagarmid. Correctness criteria beyond serializability. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 501–506. Springer, 2009.

[47] P. K. Panda, S. Swain, and P. Pattnaik. Review of some transaction models used in mobile databases. *International Journal of Instrumentation, Control & Automation (IJICA)*, 1(1):99–104, 2011.

[48] R. Pandya. *Mobile and Personal Communication Systems and Services*. IEEE Series on Digital & Mobile Communication. Wiley, 2004.

[49] E. Pitoura. A replication schema to support weak connectivity in mobile information systems. In R. Wagner and H. Thoma, editors, *Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA '96)*, pages 510–520. Springer London, 1996.

[50] E. Pitoura and B. Bhargava. Maintaining consistency of data in mobile distributed environments. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 404–413. Institute of Electrical and Electronics Engineers (IEEE), 1995.

[51] E. Pitoura and B. Bhargava. Data consistency in intermittently connected distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 11(6):896–915, November 1999.

[52] E. Pitoura and G. Samaras. *Data management for mobile computing*, volume 10. Springer Science & Business Media, 2012.

[53] G.-C. Roman, G. P. Picco, and A. L. Murphy. Software engineering for mobility: A roadmap. In A. Finkelstein, editor, *Proceedings of the Conference on The Future of Software Engineering (ICSE '00)*, pages 241–258. Association for Computing Machinery (ACM), 2000.

[54] M. Satyanarayanan. Fundamental challenges in mobile computing. In J. E. Burns and Y. Moses, editors, *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA, May 23-26, 1996*, pages 1–7. Association for Computing Machinery (ACM), 1996.

[55] W. N. Schilit. *A system architecture for context-aware mobile computing*. PhD thesis, Columbia University, 1995.

[56] P. Serrano-Alvarado, C. Roncancio, and M. Adiba. A survey of mobile transactions. *Distributed and Parallel Databases*, 16(2):193–230, 2004.

[57] P. Serrano-Alvarado, C. L. Roncancio, and M. Adiba. Analyzing mobile transactions support for dbms. In A. M. Tjoa and R. R. Wagner, editors, *Proceedings 12th International Workshop on Database and Expert Systems Applications, 2001.*, pages 595–600. Institute of Electrical and Electronics Engineers (IEEE), 2001.

[58] A. Sharma and V. Kansal. An analysis of mobile transaction methods and limitations in execution of m-commerce transaction. *International Journal of Computer Applications*, 43(21), 2012.

[59] Statista GmbH and EHI Retail Institute GmbH. Study 'Der deutsche E-Commerce-Markt 2014', 2014.

[60] R. Tewari and P. Grillo. Data management for mobile computing on the internet. In R. Brice, C. J. Hwang, and B. W. Hwang, editors, *Proceedings of the 1995 ACM 23rd Annual Conference on Computer Science (CSC '95)*, pages 246–252. Association for Computing Machinery (ACM), 1995.

[61] G. D. Walborn and P. K. Chrysanthis. Pro-motion: Management of mobile transactions. In B. Bryant, J. Carroll, D. Oppenheim, J. Hightower, and K. M. George, editors, *Proceedings of the 1997 ACM symposium on Applied computing (SAC '97)*, pages 101–108. Association for Computing Machinery (ACM), 1997.

[62] G. Weikum and G. Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Series in Data Management Systems. Morgan Kaufmann, 2002.