Information Sharing Modalities for Mobile Ad-Hoc Networks

Alexandre de Spindler, Michael Grossniklaus,
Christoph Lins, and Moira C. Norrie

Institute for Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
{despindler,grossniklaus,lins,norrie}@inf.ethz.ch

Abstract. Current mobile phone technologies have fostered the emergence of a new generation of mobile applications. Such applications allow users to interact and share information opportunistically when their mobile devices are in physical proximity or close to fixed installations. It has been shown how mobile applications such as collaborative filtering and location-based services can take advantage of ad-hoc connectivity to use physical proximity as a filter mechanism inherent to the application logic. We discuss the different modes of information sharing that arise in such settings based on the models of persistence and synchronisation. We present a platform that supports the development of applications that can exploit these modes of ad-hoc information sharing and, by means of an example, show how such an application can be realised based on the supported event model.

1 Introduction

Recent advances in mobile phone technologies have promoted the development of a new generation of mobile applications that allow users to interact and share information opportunistically based on ad-hoc network connections. Information may either be shared between mobile devices in a peer-to-peer (P2P) manner or between mobile devices and fixed installations. Both forms of information sharing have been used in different ways in a variety of applications. For example, ad-hoc network connectivity between peers has been used to detect the physical copresence of users in social settings as a basis for exchanging rating data in a P2P manner in recommender systems [1]. However, P2P connections between mobile devices have also been used as a means for storing data associated with a location based on the movement of data between peers as they pass through that location [2]. The sharing of data between mobile devices and fixed installations has been used to provide location-based services such as allowing mobile users to access local data from a server when in proximity to particular wireless hotspots or Bluetooth stations [3,4].

An analysis of the different forms of opportunistic information sharing based on ad-hoc connectivity reveals two main characteristics that determine the form of sharing, namely *persistence* and *synchronisation*. If information is shared in

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 322–339, 2009.

a persistent manner, it means that the data will be stored on the client after disconnection. Otherwise, information is only shared when the devices are connected and hence data is transient on the client. If information is shared in a synchronised manner, it means that copies of data should eventually merge so that the effects of any updates are reflected in all copies. These characteristics are orthogonal to each other, meaning that they can be combined in different ways to offer four basic modes of information sharing.

In this paper, we examine these different modalities of opportunistic information sharing in detail and present a framework that allows application developers to select and combine these in flexible ways according to the requirements of particular applications. By using a specific application scenario that exhibits all four forms of information sharing, we explain how the framework supports application development.

We start in Sect. 2 with an overview of related work on data sharing in mobile settings. Section 3 then examines the different forms of information sharing in detail using our particular application scenario to show how these can be used to meet different requirements in terms of persistence and synchronisation. Section 4 presents a platform and abstract model for the realisation of these modes based on shared data collections, while Sect. 5 shows how an application can be developed using the platform. Section 6 describes how we implemented the platform based on an extension of an existing P2P collection framework together with a unified event model for distributed object databases. We compare the resulting framework with existing platforms and frameworks for the development of mobile applications in Sect. 7. Concluding remarks are given in Sect. 8.

2 Background

Although mobile phone vendors now offer tools for the development of mobile applications, these tend to offer only basic features for data persistence and sharing. For example, the platform independent Java Wireless Toolkit (Java WTK) uses a simple key-value store for data persistence which means that developers have to define and implement a mapping between Java application objects and key-value pairs for each application. With Android[1], Google has taken another path by integrating SQLite[2] and offering the developer methods that take SQL statements as string-typed arguments similar to JDBC[3]. This results in the classic impedance mismatch problem between the application model and the storage model of data along with a lack of compile-time safety. Alternative methods are available where the single components of SQL statements can be provided individually, but this results in methods with many arguments, some of which are not used in most cases and therefore must be set to null. This still results in Java code not being checked during compilation as the table and column names are provided as string values. In contrast, development platforms for PCs such as

[1] http://code.google.com/android
[2] http://www.sqlite.org
[3] http://java.sun.com/products/jdbc

db4o[4] support Java object persistence and therefore avoid both the impedance mismatch problem and the lack of compile-time safety checks.

Support for information sharing is also limited in these platforms and data exchange must be implemented based on sockets able to send and receive binary data. Short-range connectivity such as Bluetooth or WiFi can be used to react to peers appearing in the physical vicinity, but there is no high-level support for vicinity awareness and data exchange. For example, using Java WTK, the developer has to implement two listeners, one registered for the discovery of a device and another which is notified about application-specific services discovered on a particular device.

Within the research community, the ad-hoc nature of connectivity leading to frequent and unpredictable disconnections is usually regarded as problematic and there is a lot of work on providing distributed persistence and synchronicity despite the dynamics of mobile environments [5,6,7,8]. However, a few researchers have investigated ways in which ad-hoc connectivity can be exploited as a means of detecting the proximity of users to locations, devices or other users. For example, several projects have developed location-based services that use short-range connection technologies such as WiFi or Bluetooth to provide mobile users with access to information about a location based on their proximity to that location [9,3,10,11,4]. When a user moves into the vicinity of the server fixed at that location, a connection is established and information relevant to the location can be viewed on the user's mobile device. When the user moves out of the connection range, typically that information is no longer accessible.

In previous work [1], we have shown how ad-hoc connectivity can be used to adapt collaborative filtering (CF) algorithms to mobile settings. Typical CF algorithms such as user-based filtering proceed in two steps, first selecting a set of similar users and then aggregating the ratings made by these users. In our approach, users share ratings when they are in physical proximity by means of ad-hoc connectivity between their mobile devices and recommendations are based on the aggregation of ratings stored locally on their device. The underlying assumption is that users who are physically copresent in the same social context will be similar and the more often such encounters happen, the greater the similarity. Thus, the very way in which data is shared filters that data according to user similarity and, therefore, the first step of user-based filtering to compute similar users is not required. In [1], we also report on studies carried out to validate the underlying assumption. Note that if users have exchanged ratings and then edit them afterwards when they are no longer connected, then the updates should be propagated if and when the users encounter each other again.

Both of the above examples, show how mobile applications may take advantage of the presence or absence of connectivity to share information opportunistically. The underlying physical proximity serves as a filter mechanism inherent to the application logic.

Some frameworks have been developed specifically for P2P connectivity in mobile settings including Mobile Web Services [12] and JXTA [13]. However,

[4] http://www.db4o.com

these tend to focus on lower-level forms of data exchange rather than information sharing. For example, in JXTA, the application development consists of specifying message formats and how they are processed in terms of request and response handling similar to that of service-oriented architectures. This results in a blending of the application logic typically embedded in an object-oriented data model and the collaboration logic specified based on a request-response scheme. Efforts to provide higher level abstractions of P2P networks have either focussed on the allocation and retrieval of identifiers to resources in fixed networks without considering any notion of handling [14] or they offer only a few limited collaboration primitives and lack support for vicinity awareness [15,16].

Within the database research community, a number of P2P database systems, overlay networks and middlewares have been developed, including Pastry [17], Piazza [18], PeerDB [19], Hyperion [20], P-Grid [21] and GridVine [22]. However, research efforts have tended to focus on issues of object identity, schema matching and query reformulation, distributed retrieval, indexing and synchronisation as well as transaction management. To date, there has been little effort on supporting developers of mobile applications that utilise P2P connectivity to share information opportunistically with other users in the vicinity.

3 Modes of Opportunistic Information Sharing

In this section, we examine the different modes of opportunistic information sharing in mobile ad-hoc networks by looking at an example application that features all four basic modes. The application scenario is graphically depicted in Fig. 1. The numbers in the figure are used in the following description to refer to the individual steps of a use case.

Assume a user intends to go to a movie theatre where they are a regular customer. Further, they have installed an application on their mobile phone that allows them to take advantage of the new technical facilities that this particular theatre offers to registered customers. Equipped with their mobile phone, our customer heads to the theatre (1). As soon as the user enters the building, the phone connects to a theatre server and the current movie schedule is displayed on their device (2). The programme contains all movies that will start playing within the next three hours along with the number of available seats. While browsing the movies, the number of available seats is updated in real time. The programme is stored persistently on the device and kept up to date by the theatre server as long as the user is connected.

While our customer is trying to decide on a movie, an advertisement suddenly pops up informing them that they have won a 50% reduction voucher for a particular movie (3). Such pop-ups are only available on the device while it is connected to the server and may be updated by the server, for example in order to upgrade from a 50% reduction voucher to a free ticket as the start time approaches. Having decided to accept the voucher, our customer orders a ticket for the proposed movie and the ticket is sent to their phone (4). The ticket data will be stored on the phone persistently, and is handled as a stand-alone piece of data not kept in sync with any data residing on the server.

Now our customer realises that they have left their spectacles at home and decides to return home to collect it before the start of the movie. When they leave the building, their application indicates that it is no longer connected with the server (5). As a consequence, the previously received advertisements vanish. In contrast, the ticket along with the list of movies remains reminding them of the start time. They later return and watch the movie (6). Their ticket gets marked as expired (7) and therefore will be removed from their mobile phone when leaving the movie theatre (8).

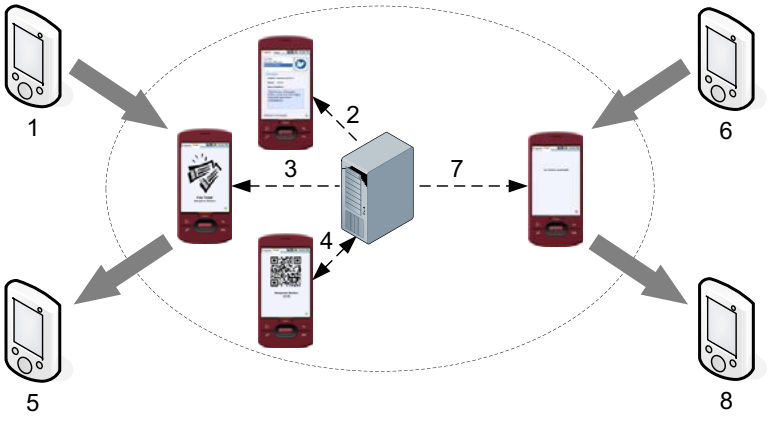

Fig. 1. Application

The short-range connections between personal mobile devices or between mobile devices and fixed installations described in this scenario allow for information to be shared based on physical proximity. In a mobile environment, connections are likely to be short-lived as devices move in and out of range. Therefore, one consideration is whether shared data should still be available when the connection is lost or if it should only be available during the connection. For example, the movie schedule that users receive as soon as they enter the theatre is stored persistently so that they can still access it after leaving the theatre. In contrast, pop-up advertisements are only available on the mobile device while the users are in the theatre. Once they have left, these advertisements are automatically removed from the mobile device.

Furthermore, in some cases, shared information should be kept synchronised to whatever extent is possible after devices are disconnected, while in other cases the copies of data should be decoupled. In our scenario, the movie schedule along with the number of available seats displayed on the mobile device is continuously updated by the theatre server. Changes carried out on the server are automatically propagated to the mobile devices. When a customer leaves the theatre, their mobile device is no longer connected and hence the copy of the movie programme stored on their device will not reflect the most recent updates. However, the next time that the customer enters the theatre and connection is

re-established that data will be updated immediately. In contrast, a movie ticket is copied to the mobile device in such a way that it is decoupled from the original object residing on the server.

Taking these cases into consideration, two orthogonal concepts can be identified. Data *continuity* refers to the question of whether the lifetime of an object on a client is limited to the lifetime of the connection. Data *coupling* addresses the question of whether objects on different devices are kept synchronised. The composition of these concepts defines four modalities of information sharing as shown in Fig. 2. The horizontal axis covers the data coupling concept and the vertical axis represents the concept of data continuity. In this figure, we assume two devices are connected and take on the perspective of information being sent from a local to a remote device.

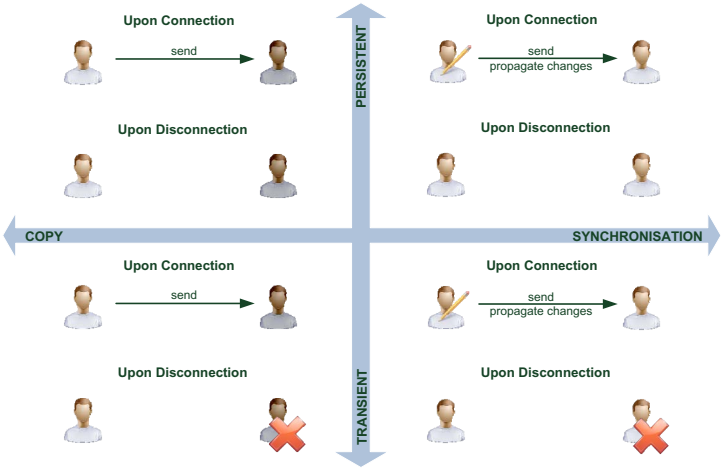

Fig. 2. Sharing Modes

The *persistent copy* mode is regarded as the default mode because it is the simplest one and is likely to be the one used most often. Upon connection, objects are copied to the remote device. The copied objects are not linked to the original ones. Instead, the original and the remote data will be treated as two distinct objects that can be manipulated independently. After disconnection, the received data remains visible allowing further offline access.

In *transient copy* mode, received objects are deleted as soon as the devices disconnect. Access to information on the receiver's device is therefore determined by the state of connectivity. However, the received objects are still treated as independent and, therefore, while the devices are connected and the data is visible, updates are not propagated.

In *persistent synchronisation* mode, when devices connect, data is transferred in such a way that it remains available after disconnection. As long as the connection is established, updates are propagated. Clearly, object synchronisation

cannot take place after disconnection. However, data will be synchronised as soon as the connection is re-established. The access to local data is always possible, hence allowing offline operation.

Finally, the *transient synchronisation* mode covers the case when data on the receiver device is only visible during connection and updates are propagated. Manipulations of the data do not need to be tracked after disconnection because, upon disconnection, the data on the remote device no longer exists.

Note that the mode in which objects are shared may be altered after they have been shared. For example, objects shared in transient mode can be set to be persistent by the receiver. As a result, these objects will not be deleted upon disconnection. Persistent objects may be set to be transient in which case they will be deleted. Similarly, synchronised objects can be decoupled and objects that have been shared in copy mode may be set to be synchronised.

4 Platform for Opportunistic Information Sharing

To support the development of mobile applications that take advantage of ad-hoc network connections for opportunistic information sharing, we have developed a general platform based on an abstract model of information sharing that offers the four basic modes of information sharing presented in the previous section. The model is based on the classification of objects into collections that determine the basic sharing mode. Further, an event model allows handlers to be registered for particular events such as the addition/removal of an object to/from a collection or updates to attribute values of objects within a collection. Upon such events, the registered handlers are notified and, consequently, the handlers execute their actions which implement the specific sharing logic.

All sharing modes are implemented based on the collections shown in Fig. 3. The root collection named `Objects` contains all existing objects and the `Shared Objects` collection is a subcollection that contains all objects that have either been sent or received. In the previous section, we identified two orthogonal sharing mode dimensions, namely transient—persistent and copy—synchronisation. Each of these dimensions is represented as a single subcollection of the `Shared Objects` collection. Thus, the `Transient` collection represents those objects which are shared and not persistent while the `Synchronised` collection represents those objects which are shared and synchronised i.e. not decoupled as in the case of copy. Objects can belong to neither, one or both of these subcollections, thereby representing the four different modes of information sharing.

Handlers registered to be notified about the arrival or departure of peers to and from physical proximity, as well as for changes to object attribute values, drive the sharing process in terms of actions. The actions consist of executing queries, propagating updates, sending objects and adding or removing sent and received objects to and from collections. The sequence of actions specific to particular sharing modes are summarised in Tab. 1.

We now outline how the collections are used in order to realise the four different sharing modes. For the sake of simplicity, we assume two peers sharing

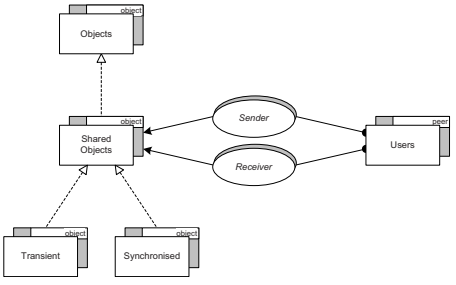

Fig. 3. Collection model of the sharing modes

Table 1. Operations associated with sharing modes

	Sender	**Receiver**
Persistent Copy	**Upon connection**	
	Insert into Shared Objects	Insert into Shared Objects
	Insert into Receiver	Insert into Sender
	Upon disconnection	
	—	—
Transient Copy	**Upon connection**	
	Insert into Shared Objects	Insert into Shared Objects
	Insert into Receiver	Insert into Sender
		Insert into Transient
	Upon disconnection	
	—	Delete
Persistent Synchronisation	**Upon connection**	
	Insert into Shared Objects	Insert into Shared Objects
	Insert into Receiver	Insert into Sender
	Insert into Synchronised	Insert into Synchronised
	Handler on synchronised objects	Handler on synchronised objects
	Propagate/Apply changes	Propagate/Apply changes
	Upon disconnection	
	Track changes	Track changes
Transient Synchronisation	**Upon connection**	
	Insert into Shared Objects	Insert into Shared Objects
	Insert into Receiver	Insert into Sender
	Insert into Synchronised	Insert into Transient
	Handler on synchronised objects	Insert into Synchronised
	Propagate/Apply changes	Handler on synchronised objects
		Propagate/Apply changes
	Upon disconnection	
	Remove from Synchronised	Delete

a single object under a particular sharing mode, where one of them acts as a sender and the other one as a receiver. One-to-many and many-to-one object sharing which typically occur in mobile settings can always be represented by multiple and directed one-to-one object transfers. Note that, for a particular

object to be shared, the sharing mode is defined and known to the system on both the sender and receiver side. The way in which objects are associated with a particular sharing mode will be explained in the following section when we show how applications are developed.

Figure 4 highlights those collections and associations to which a shared object or other objects are added and how they are associated. In the four sharing modes, all sent and received objects are inserted into the `Shared Objects` collection. Additionally, received objects are associated with the sending peer using the `Sender` association on the receiver side. Thus, we can avoid exchanging objects that have already been sent in persistent sharing modes. Conversely, sent objects are associated with the receiving peer using the `Receiver` association on the sender side. This information is exploited for the propagation of updates.

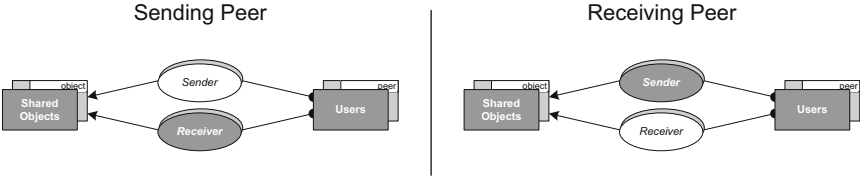

Fig. 4. Association of shared objects with sending and receiving peer

Figure 5 shows the collections to which an object being shared is added. The set of collections to which it is added can be seen as a configuration resulting in the object being shared in a configuration-specific sharing mode. The configuration is a classification and therefore it is the classification that determines the mode. Since objects can be dynamically classified at runtime, the sharing mode can also be managed dynamically at runtime.

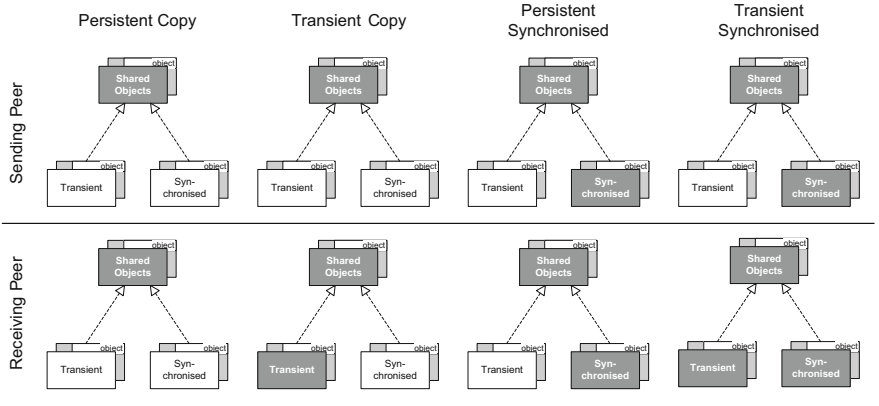

Fig. 5. Sharing mode-specific classifications of shared objects

If an object is to be shared in persistent copy mode, it is not classified any further. As a result, the object will simply be sent or received, persist despite disconnections and not be kept synchronised. If an object is shared in transient mode, it will additionally be put into the `Transient` collection on the receiver side, when it is received. Upon disappearance of the sending peer, a handler retrieves all members of this collection which are associated with the disappearing peer by means of a query. These objects are deleted in order to guarantee that they are no longer visible nor accessible.

In the case that an object should be shared in synchronised mode, it is put into the `Synchronised` collection on the sender and receiver sides, before being sent and after having been received, respectively. On both sides, a handler is registered to observe changes on all members of that collection. When updates are performed, the handler action will propagate the changes to the peers associated with the updated objects. If such peers are not in the vicinity, the handler retains its action so that it will be re-executed as soon as a connection has been re-established. Finally, if an object is to be shared in transient synchronised mode, it will be put into the `Synchronised` collection on both the sender and receiver sides as well as into the `Transient` collection on the receiver side. The effect will be the combination of the effects of having the object in one of the collections as described for the persistent synchronised and transient copy mode.

The sharing mode of an object is changed by adapting its classification. An object configured to be shared in transient mode can be removed from the `Transient` collection during connection. As a result, it will not be deleted when the connection is lost. Conversely, persistent objects may be added to this collection in order to make them transient. The synchronisation mode may be altered by adding or removing objects from the `Synchronised` collection.

5 Application Development

Having presented our model of the sharing modes based on collections, we now describe the implementation of the application scenario presented in the previous section. For the sake of a comprehensive overview, we start at the beginning by presenting how a database is created and opened using the following code.

```
DatabaseManager dbms = new DatabaseManager();
dbms.createDatabase("MovieTheatreApplication");
Database db = dbms.open("MovieTheatreApplication");
```

The application domain is modeled by four different types which are depicted in Fig. 6 by means of UML classes. Since these types are implemented as regular Java classes, we do not show the program code. The `Movie` class consists of a name, a start time, number of available seats and a description. A `Ticket` has attributes referencing the customer peer and the movie. A peer object contains all necessary information to identify and connect to a peer. An `Order` contains a peer and a movie object. In terms of advertisements, an image together with a string phrase can be specified when an object of type `Advertisement` is created.

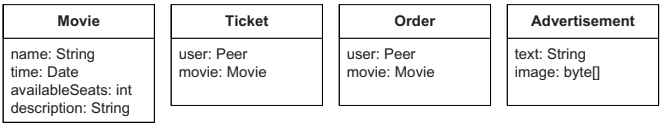

Fig. 6. UML diagrams of domain types

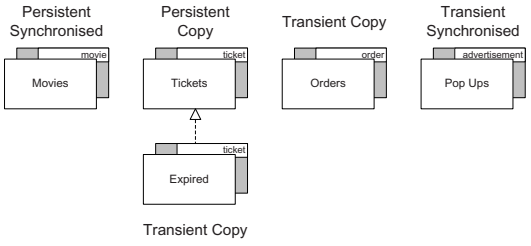

Fig. 7. Collection model

As shown in Fig. 7, a collection for every type is created to manage the extents of the types as well as one subcollection to classify tickets as expired. These collections are all set to be persistent collections with the effect that all members are stored in the database. A collection is created by providing its name and membertype and made persistent as shown by the following statements.

```
Collection<Movie> movies = db.createCollection("Movies", Movie.class);
movies.setPersistent();
```

When the user in our scenario enters the building for the first time, their mobile device automatically connects to the server provided by the movie theatre. The current movie programme is presented to the user based on the following query to select movies showing in the next 3 hours:

```
QueryNode<Movie> selection = db.queries().
  select(movies, Movie.time, System.currentTimeMillis()
    + 3 * 3600000, BinaryOperator.SMALLER);
```

The application running on the handheld device makes the `Movies` collection available and therefore it is able to receive members. Available collections contain those objects that are shared and are made available in a particular sharing mode. Therefore, the sharing mode for a particular object is defined by the collection through which it is shared. In the case of the `Movies` collection, whenever changes to the movie items are carried out on the server, the updates are forwarded to all visitors. This is realised by making the movie collection available in persistent synchronised mode as shown in the following code:

```
movies.makeAvailable(selection, Synchronised.TRUE && Persistent.TRUE);
```

The sharing is initiated by the appearance of a peer in physical proximity to the movie theatre which is translated to a new member added to the `Vicinity` collection. Consequently, a handler must be created which executes the query

specified above and sends the query result to the new peer in the vicinity. This handler is then registered with the vicinity collection for addition events.

```
class VicinityHandler implements Handler {
  private Collection collection;
  public VicinityHandler(Collection collection) {
    this.collection = collection;
  }
  public void action(Event event) {
    Peer peer = ((AddEvent)event).getMember();
    Collection<Movie> result = this.collection.query().execute();
    Connectivity.send(peer, result);
  }
};
Handler handler = new VicinityHandler(movies);
db.queries().collection("Vicinity").addAdditionHandler(handler);
```

The other collections are created and made available similarly. The `Advertisements` collection is made available on the device of the user as well as on the server. As long as the user resides within the building and stays connected, advertisements can be distributed and updated at any time as defined for the transient synchronised mode.

The functionality to order a ticket is also implemented based on the notion of shared collections. The `Orders` collection is made available on both the mobile devices and the server. The server application acts as a client and receives order items. Since these items are only required to exist while the user is at the cashpoint, the availability is set to transient. The receipt of an order item will be treated by a handler registered for addition events on the `Orders` collection. The handler action is executed resulting in the creation of a ticket object that is added to the `Tickets` collection which will be transfered directly to the requesting device. Orders are handled immediately which leaves no time for updates. Therefore this collection is not set to be in synchronisation mode.

A ticket is a unique item that cannot be changed by the client nor by the server application. It stays visible independently of whether a connection is established or not. Both of these requirements are covered by the persistent copy mode. However, as soon as a movie starts, the server application sets the user's ticket to be expired. This is achieved by adding the ticket object to the subcollection of the `Tickets` collection named `Expired`. This subcollection has been made available with a transient copy sharing mode which overrides the sharing mode of the supercollection. Therefore, the ticket object residing on the mobile device will be removed from the device as soon as the user leaves the theatre.

6 Platform Implementation

Our platform is composed of three components as shown in Fig. 8. A collection framework allows objects to be classified, shared and stored persistently. An event system offers the means to define events, to register handlers to be notified

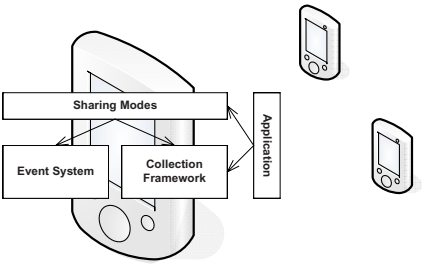

Fig. 8. Architecture

about events and to associate actions with handlers. Finally, a sharing mode component implements the modalities introduced in Sect. 3 based on the other two components.

An application programming interface (API) provides access to a database management system that manages collections, events, handlers and query representations. Through the API, database instances can be created, opened, closed and deleted, as well as allowing objects, collections, events, handlers and queries to be created, retrieved and deleted, defined and executed with a database instance. The platform is implemented in Java allowing mobile applications to be implemented using the regular Java platform.

6.1 Collections

Our platform is based on an extension of a P2P collection framework that we developed previously [23]. We therefore first give an overview of the concept of a P2P collection. Fig. 9 presents the concept as a UML diagram. A P2P collection has a name with which it can be retrieved within a query and membertype defining the type of its members. It allows members to be accessed and dynamically added and removed at runtime. A P2P collection may optionally store its members persistently which can be configured at design and runtime. Furthermore, a query system allows P2P collections to be retrieved, unified, intersected and objects to be selected, mapped and accessed.

As shown in Sect. 5, sharing is implemented by making collections available or unavailable. An available P2P collection can send and receive members to and from compatible collections residing on remote peers. Two P2P collections are compatible if they have the same name and compatible membertypes. Members may be sent upon explicit request or automatically as a result of a peer appearing in physical proximity to another one. Members that are sent from a source collection to a target collection will appear as members of the target collection. In some cases where members are shared automatically as soon as users are in each other's proximity, there must be a way of selecting which members of a collection to send. For this purpose, a query can be defined and provided when the P2P collection is made available to return the required members.

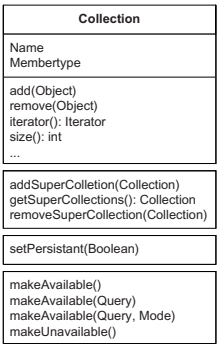

Fig. 9. Collections

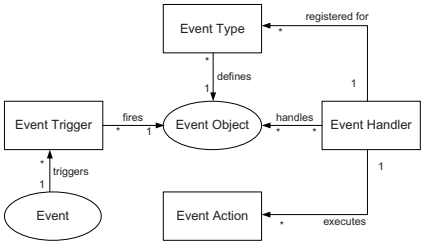

Fig. 10. Event concepts

Having P2P collections as a central point for information sharing affects the manner in which applications are developed. In a model-driven development scenario, this results in one additional step. First, the application domain is modeled in terms of types which map to Java classes. In a second step, P2P collections are defined. By default, a P2P collection is created for each domain type to represent its extent. Collections and subcollections as well as associations may be added at any time in order to further classify domain objects.

6.2 Events

The second component of our platform for information sharing in mobile ad-hoc networks is an event system based on a unified event model for object databases [24,25]. We describe here those concepts required in the scope of this work. The original motivation of our event system was to unify concepts for event handling found in object-oriented programming languages with trigger mechanisms from database systems in the setting of object databases. Current object-oriented databases offer limited support for event handling and often delegate this issue to the programming language. We believe that object databases need a well-defined event model that offers the generality and flexibility supported in active databases and modern event processing systems.

We base our event model on the four concepts of *event triggers*, *event types*, *event handlers* and *event actions* shown in Fig. 10. Event triggers are an abstraction of low-level operations that occur in the database such as an object being created, changed or added to a collection. Apart from being bound to such database operations, event triggers can also be bound to a schedule that determines when they are fired. Finally, they can also be fired explicitly by the user or the application code. Each event trigger is associated with an event type that defines the type of the event objects that are fired by the trigger. In addition to an attribute identifying the source of the event, event objects can also carry other information depending on the context in which they were fired. Whenever an event object is created by a trigger, our system matches it to event handlers that are registered to process events of a given type. All matching handlers are

notified and the event object is passed to them. A handler can define a conditional expression that guards the execution of the event handling logic. If this expression evaluates to true, the handler invokes its associated event action and passes the required parameters.

By introducing separate concepts for event triggers, types, handlers and actions, event processing functionality can be made orthogonal to persistence. The model supports user-defined as well as system-defined event types with a registration service that allows one or more handlers to be associated with the same event type. Further, the model supports reuse of event types and event distribution which is an important requirement in the context of this work.

6.3 Sharing Modes

Once a P2P collection has been made available, it can either share members by explicit request or automatically upon the arrival of a peer in the physical vicinity. In the latter case, the collection is notified about the new peer, executes the query and sends the query result with the sharing mode specified by the available collection. As part of the P2P collection framework, a system collection named `Vicinity` contains objects representing peers. This collection is a direct representation of a peer's physical proximity and contains all those peers that are in the vicinity. A new peer appearing in the vicinity will trigger the creation of a new peer object which will be added to this collection. Conversely, the disappearance of a peer will trigger the removal of the respective object from the vicinity collection. Available collections register handlers with this vicinity collection to be notified about the addition or removal of peer objects. Upon notification, the mode-specific sharing process is executed.

Our platform for opportunistic information sharing integrates the P2P collection framework with the event system and also introduces new functionality to implement the sharing modes. By default, an available collection shares its members in persistent copy mode. If another mode is favoured, a particular mode can be provided as a second argument when the collection is made available. As a result, the sharing mode for a particular object is defined by the available collection of which it is a member. A sharing mode may be changed at any time by making a collection available once more with the new mode. Note that sharing modes of subcollections override the modes of the supercollections, only for those objects that are members of the subcollections. As a result, not only can the sharing mode of an available collection be changed dynamically at runtime but, using subcollections, it can also be adapted for dynamic subsets of objects.

7 Discussion

In [23], we compare the effort required to implement mobile applications using our P2P collection framework with that required if using mobile phone SDKs such as Java WTK. Figure 11 compares the components needed and the amount of interaction required to implement data persistence, vicinity awareness and

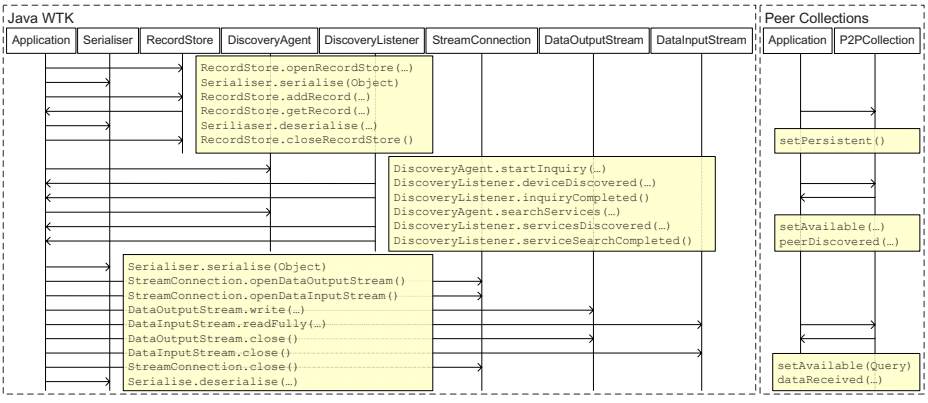

Fig. 11. Comparison of using Java WTK (left) and our platform (right)

data sharing using Java WTK, on the left hand side, and our P2P collection framework, on the right hand side. The ability to be able to simply set collections to store their members persistently is a great improvement compared to Java WTK where application objects have to be serialised manually and stored using key-value records. In order to store objects of a particular type based on key-value pairs, an application developer has to program a database-like component and put a lot of effort into overcoming the impedance mismatch between objects and key-value pairs. As opposed to the simple vicinity awareness mechanism provided by our platform, the developer of a Java WTK application needs to implement the scanning of the environment based on low-level connection technologies such as Bluetooth or WiFi. Moreover, using our framework, the application developer does not have to bother with low-level socket-based connectivity and data transmission in terms of serialisation and deserialisation. The fact that the developer can work at the level of the application model by deciding how data collections should be shared, and in which mode, presents a significant contribution to the development of mobile applications.

If a developer manages to implement data persistence, vicinity awareness and data sharing, the resulting sharing mode will be persistent copy. If another mode is required, the developer needs to additionally implement the mode-specific behaviour as described in Tab. 1. For transient mode, shared objects must be deleted from the device as soon as a peer is no longer in the vicinity, while the synchronisation of objects requires objects to be observed and updates to be conveyed and applied. The fact that our platform does all of this greatly simplifies the developer's work as they are simply required to select sharing modes rather than having to implement them.

8 Conclusions

Mobile applications with functionality to share information in ad-hoc peer-to-peer networks are becoming increasingly popular. These applications target

domains such as collaborative filtering and location-based services which exploit the ad-hoc connectivity as a means of sharing information opportunitistically, thereby filtering information based on proximity. Therefore, in this novel class of applications, connections and disconnections throughout the life-time of a program are an integral part of the application logic rather than something to be hidden from the user and developer alike.

Due to the limited support for opportunistic information sharing in existing development platforms for mobile applications, we have designed and implemented a platform that provides high-level support to application developers. Central to the platform is support for the four basic modes of opportunistic information sharing that we have identified. These modalities are based on the observation that ad-hoc information sharing is mostly governed by two main factors, namely persistence and synchronisation. This platform builds on a P2P collection framework for sharing and persistence, an event system for the specification of event-driven application behaviour and a component that provides support for the four different sharing modes. We have shown how applications can make use of the functionality offered by the platform and compared it to the effort required using an off-the-shelf development kit.

References

1. de Spindler, A., Norrie, M.C., Grossniklaus, M.: Recommendation based on Opportunistic Information Sharing between Tourists. Information Technology & Tourism 10(4) (2009)
2. Xu, B., Ouksel, A., Wolfson, O.: Opportunistic Resource Exchange in Inter-Vehicle Ad-Hoc Networks. In: Proc. Intl. Conf. on Mobile Data Management, MDM (2004)
3. Burrell, J., Gay, G.K.: E-graffiti: Evaluating Real-World Use of a Context-Aware System. Interacting with Computers 14(4) (2002)
4. Fernandez, C., Escudero, C., Iglesia, D., Rodriguez, M., Marcote, E.: MOVILTOOTH: a Bluetooth Context-Aware System with Push Technology. In: Proc. IEEE Mediterranean Electrotechnical Conference, MELECON (2006)
5. Park, I., Hyun, S.J.: A Dynamic Mobile Transaction Management Strategy for Data-intensive Applications based on the Behaviors of Mobile Hosts. In: Information Systems and Databases (ISDB), pp. 92–97 (2002)
6. Repantis, T., Kalogeraki, V.: Data Dissemination in Mobile Peer-to-Peer Networks. In: Proc. Intl. Conf. on Mobile Data Management, MDM (2005)
7. Lee, M., Helal, S.: HiCoMo: High Commit Mobile Transactions. Distrib. Parallel Databases 11(1) (2002)
8. Padmanabhan, P., Gruenwald, L., Vallur, A., Atiquzzaman, M.: A Survey of Data Replication Techniques for Mobile Ad Hoc Network Databases. The VLDB Journal 17(5) (2008)
9. Leonhardi, A., Kubach, U., Rothermel, K., Fritz, A.: Virtual Information Towers – a Metaphor for Intuitive, Location-Aware Information Access in a Mobile Environment. In: Proc. IEEE Intl. Symp. on Wearable Computers, ISWC (1999)
10. Cano, J.-P., Manzoni, P., Toh, C.K.: First Experiences with Bluetooth and Java in Ubiquitous Computing. In: Proc. IEEE Symposium on Computers and Communications, ISCC (2005)

11. D'Souza, M.J., Postula, A.J., Bergmann, N.W., Ros, M.: Mobile Locality-Aware Multimedia on Mobile Computing Devices; A Bluetooth Wireless Network Infrastructure for a Multimedia Guidebook. In: Proc. Intl. Conf. on E-Business and Telecommunication Networks, ICETE (2005)
12. Srirama, S.N., Jarke, M., Prinz, W.: Mobile Web Services Mediation Framework. In: Proc. Workshop on Middleware for Service Oriented Computing, MW4SOC (2007)
13. Traversat, B., Arora, A., Abdelaziz, M., Duigou, M., Haywood, C., Hugly, J.C., Pouyoul, E., Yeager, B.: Project JXTA 2.0 Super-Peer Virtual Network. Technical report, Sun Microsystems, Inc. (2003)
14. Aberer, K., Alima, L.O., Ghodsi, A., Girdzijauskas, S., Haridi, S., Hauswirth, M.: The Essence of P2P: A Reference Architecture for Overlay Networks. In: Proc. IEEE Intl. Conf. on Peer-to-Peer Computing, P2P (2005)
15. Wang, A.I., Bjornsgard, T., Saxlund, K.: Peer2Me – Rapid Application Framework for Mobile Peer-to-Peer Applications. In: Intl. Symp. on Collaborative Technologies and Systems, CTS (2007)
16. Kortuem, G., Schneider, J., Preuitt, D., Thompson, T.G., Fickas, S., Segall, Z.: When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks. In: Proc. Intl. Conf. on Peer-to-Peer Computing, P2P (2001)
17. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, p. 329. Springer, Heidelberg (2001)
18. Tatarinov, I., Ives, Z., Madhavan, J., Halevy, A., Suciu, D., Dalvi, N., Dong, X.L., Kadiyska, Y., Miklau, G., Mork, P.: The Piazza Peer Data Management Project. SIGMOD Rec. 32(3) (2003)
19. Ooi, B.C., Tan, K.L., Zhou, A., Goh, C.H., Li, Y., Liau, C.Y., Ling, B., Ng, W.S., Shu, Y., Wang, X., Zhang, M.: PeerDB: Peering into Personal Databases. In: Proc. ACM SIGMOD Intl. Conf. on Management of Data, SIGMOD (2003)
20. Rodríguez-Gianolli, P., Kementsietsidis, A., Garzetti, M., Kiringa, I., Jiang, L., Masud, M., Miller, R.J., Mylopoulos, J.: Data Sharing in the Hyperion Peer Database System. In: Proc. Intl. Conf. on Very Large Databases, VLDB (2005)
21. Aberer, K., Datta, A., Hauswirth, M., Schmidt, R.: Indexing Data-Oriented Overlay Networks. In: Proc. Intl. Conf. on Very Large Databases, VLDB (2005)
22. Cudré-Mauroux, P., Agarwal, S., Budura, A., Haghani, P., Aberer, K.: Self-Organizing Schema Mappings in the GridVine Peer Data Management System. In: Proc. Intl. Conf. on Very Large Databases, VLDB (2007)
23. de Spindler, A., Grossniklaus, M., Norrie, M.C.: Development Framework for Mobile Social Applications. In: Proc. Intl. Conf. on Advanced Information Systems Engineering, CAiSE (2009)
24. Grossniklaus, M., Norrie, M.C., Sgier, J.: Realising Proactive Behaviour in Mobile Data-Centric Applications. In: Proc. Intl. Workshop on Ubiquitous Mobile Information and Collaboration Systems, UMICS (2007)
25. Grossniklaus, M., Leone, S., de Spindler, A., Norrie, M.C.: Unified Event Model for Object Databases. In: Proc. Intl. Conf. on Object Databases, ICOODB (2009)