A Platform for Supporting Micro-Collaborations in a Diverse Device Environment

doi:10.3991/ijim.v3i4.1039

David Johnson
University of Reading, Reading, United Kingdom

***Abstract*—Collaborative software is usually thought of as providing audio-video conferencing services, application/desktop sharing, and access to large content repositories. However mobile device usage is characterized by users carrying out short and intermittent tasks sometimes referred to as "micro-tasking". *Micro-collaborations* are not well supported by traditional groupware systems and the work in this paper seeks out to address this. Mico (pronounced *mee-koh*) is a system that provides a set of application level peer-to-peer (P2P) services for the ad-hoc formation and facilitation of collaborative groups across a diverse mobile device domain. The system builds on the Java Micro Edition bindings of the JXTA P2P protocols, and is designed with an approach to use the lowest common denominators that are required for collaboration between varying degrees of mobile device capability. To demonstrate how our platform facilitates application development, we built an exemplary set of demonstration applications and include code examples here to illustrate the ease and speed afforded when developing collaborative software with Mico.**

***Index Terms*—mobile, peer-to-peer, collaborative work, group communications software**

I. INTRODUCTION

With the advances in computing technology and the ever falling costs of computer hardware mobile devices are becoming commonplace in all aspects of life. The increasing availability of wireless networks has contributed to this growth as many mobile devices are wireless enabled. Consumer electronics such as mobile phones and personal digital assistants (PDAs) are now being used as mobile computing devices as the hardware capability of such devices now allows for open application development that is not restricted by the vendor or manufacturer. Wireless connectivity has allowed mobile devices to connect to the Internet in not just a consumer capacity, but also as information providers.

Wiberg [1] describes the field of Mobile Computer Supported Cooperative Work (Mobile CSCW) as when people use mobile computing devices for collaboration, where each user is not fixed by location and the users are geographically dispersed. This should not be confused with telework where people work at distance from their usual workplace and a worker may still be constrained to working in a single place. Kristoffersen and Ljungberg [2] identified three modalities of mobile work: wandering, travelling, and visiting. *Wandering* is working whilst being mobile locally (local to other mobile users). One such example may be a team of IT support staff each using mobile devices to coordinate their activities. *Travelling* is working whilst going from one place to another in some form of transport, or travelling some significant distance. *Visiting* is working in different places for a relatively short amount of time. For example, a building surveyor might visit several different sites throughout a day's work.

These different modes of mobile work bring new challenges to how a distributed system would operate with mobile computing devices. Compare Kristoffersen's notions of wandering and visiting. A distributed system that supports *wandering* may involve communication between local mobile users. This can be supported with a wireless Local Area Network, or even by creating a pico-net supported by Bluetooth connectivity. A system that support *visiting* may involve wholly disconnected nodes for the duration of a site visit, or short tasks carried out at multiple locations without direct communication with another user. Within each of these broad subsets of mobile work, the way in which users perform activities differ significantly from the traditional desktop user.

Many factors have a direct impact of how mobile software is design and there has been much research into how to increase productivity when using mobile devices. Brandt et al [3] from the Stanford University HCI Group surveyed a range of applications that can be used to manage a user's tasks including Microsoft OneNote, Google Calendar, and Facebook. They also include in their survey two of their own applications, *4l8r* and *ButterflyNet*, that are specifically designed around the notion of assisting users with limited attention. The authors coin the term *tasklet* to describe "a small portion of an activity undertaken in situations characterized by limited available attention." They go on to discuss five aspects to consider for designing for such users.

Variation in the value of time: For a given task, a mobile user is not always able or willing to carry out the task in the immediate context in which it prompts attention. By analyzing usage patterns of their 4l8r mobile application, the authors found that users performed tasklets of recording reminders of events throughout the day, and completed the full activity of writing diary entries in a more convenient context, usually with less constraint on their time.

Availability of information in context: Storing and retrieving information becomes particularly important when in time constrained situations. Being able to quickly record information, including the context in which it was saved, allows information systems to be built that can be organized by user context. Retrieving the information can be made easier by automatically organizing information by time, location, and through user-defined context

descriptions. The authors identified that some tasks are time-sensitive, and information can be organized for a user to be readily presented with the most contextually relevant data.

Social dynamics of tasklets: When activities involve multiple people, social elements have to be considered. Collaboration requires coordination, communication, and sharing within a group. The authors observed that, particularly in social networking applications, users rarely take into consideration privacy issues when segmenting activities with tasklets. For example, when micro-coordinating, a typical tasklet would be to notify a group of people of an as-yet unplanned event. Although not all recipients of the notification are known to want to participate in the tasklet (i.e. they may not want to be notified), the value of time is such that the burden of responsibility is pushed to the whole group. The tasklet of removing oneself from the micro-coordination is seen as a "cheap" enough to be acceptable practice in this kind of micro-coordination.

Functionality versus Complexity: There are a range of factors to consider in designing any software application, each of which has direct influence on others. However the authors discuss in particular the tradeoffs between supporting multiple modes of interaction and how complex each mode is to use. They found that having a choice of modality is important to users, and the mode is not determined by context as hypothesized, but simply by user preference. The level of automation is also an issue, where it is almost impossible to design for all behaviours without having a structured and standard style of input. An overriding problem with automation is with how a user proceeds when automation fails. It is not always straightforward to find a good balance between simplicity for the user and encapsulating complexities with higher-level functionality.

Levels of feedback: Appropriate feedback is essential in any activity, however the authors identify that it is not always necessary or desirable for feedback in all cases. With tasklets, feedback can be cumbersome and interrupt a user's workflow, where the flow might augment with tasklets outside of the software system and even outside of the device itself (i.e. into the "real-world"). Within the domain of tasklets, different tasklets might hold different values to the user, a feature the authors also noted when comparing evaluations of 4l8r and ButterflyNet.

One must note about each of the factors is that there is an assumption that each application surveyed is ubiquitously accessible. Many of these applications are Web-based, with the WWW being a relatively pervasive computing platform. However Roth [4] identified the problem of diversity when considering mobile devices. The network protocols for communication, the operating system, and the hardware itself varies as the spectrum of consumer mobile electronics is broad, more so than with desktop platforms. The way to tackle diversity is by adopting standards. If each and every device in a distributed network can interpret the same machine code, and understand the same network protocols, then only one version of software would ever have to be compiled and deployed ubiquitously. This reality however has not yet been realized.

We submit that tasklets are a central theme to activity-based mobile computing, where micro-tasking is a theme that arises time and again in various works describing mobile CSCW research. However researching the collaborative aspects of designing for tasklet based interactions still has a lot to be desired. This paper describes our overarching aim of producing a software library for micro-collaboration application development that attempts to overcome the problems associated with the diversity in mobile computing and placing an emphasis on issues surrounding tasklet based collaborations.

II. THE MICO MICRO-COLLABORATION PLATFORM

To tackle the issues surrounding heterogeneous devices and networks, we build on existing technologies that address different aspects of diversity in the mobile device domain. To target a generalized mobile software platform, we use Java Micro Edition as our underlying platform agnostic mobile application environment. To aid in handling unpredictable network configurations, we build on the platform independent JXTA protocols to form our collaborative groups.

A. Foundation Technologies

Project JXTA aims to enable the creation of networked services and applications through a standard set of protocols to allow computing devices to provide and consume services, and to share resources. JXTA provides a basic framework and set of services that allow peers to form self-organized groups in a heterogeneous network environment without the need for any central mediation or management. Some of the key objectives of JXTA as described by Gong [5] are described as follows.

Interoperability: JXTA provides a standard set of open XML-based network protocols to facilitate peer-to-peer (P2P) networking. When the project was originally conceived, there was much fragmentation in the P2P protocol domain. Although P2P applications and systems were and still are prevalent, and account for a significant portion of all Internet traffic, each of the different systems developed uses its own proprietary protocol that is usually application-specific. Project JXTA aims to overcome this by providing an open and standardized protocol for highly distributed computing.

Platform Independence: Apart from the diversity found in the P2P application domain, a wider problem exists in that there is a diverse ecosystem of computing architectures, software programming languages, and networking technologies. For example, one particular system might provide an environment which favours the development of applications on the Mac OSX operating system written in the Objective-C programming language, with network access over TCP/IP, while another might only provide HTTP as a transport protocol in a Java-based programming and operating environment. If a developer wishes to serve both communities, significant effort is required to be able to develop on both platforms and to build a bridge between the two systems. JXTA is designed to be independent of platform-specific elements and as such seeks to be all embracing.

Ubiquity: A range of devices are prevalent in modern distributed systems, and it is important to be able to design applications that can interoperate between disparate kinds of hardware. Many distributed systems are designed in a rigid fashion usually targeting a single class of device,

such as desktop PCs, or follow the client-server paradigm where an enterprise server exclusively provides the service to end-user client software. With a paradigm where all devices are expected to provide and maintain decentralized group services, JXTA looks to be a technology that encompasses all kinds of computing device including traditional enterprise servers and desktop PCs, to small devices such as mobile phones and smart cards. As a distributed technology enabler for any device, JXTA provides an easier way to develop networked services and applications that spans device boundaries.

Building on the features provided by Java Micro Edition (Java ME) means that we can target a platform environment that spans a range of capability device. Targeting the Connected Limited Device Configuration and Mobile Independent Device Profile (CLDC/MIDP) platform combination encompasses low capability mobile devices up to high-end smartphones and PDAs without having to put in a large amount of effort in developing the software to run across a broad range of devices. JXTA having existing reference implementations for Java ME means that JXTA can be used as the enabling technology that builds a virtual overlay network on top of underlying heterogeneous network infrastructures. With JXTA taking care of enabling communication across multiple types of network, and Java ME providing a platform capable of supporting application development across multiple kinds of mobile devices, we builds on these technologies to provide a library of services that allow the easy development of collaboration applications and services.

B. Architecture

Typically groupware is designed around a set of common concepts, with different developers rehashing the same core ideas. The concepts described in this section are not necessarily novel, nor are they trivial as their definition forms the basis for our collaborative system. We designed Mico around the following core concepts: *Communication*, *Content*, *Users* and *Groups*.

Communication: At least the most basic form of communication between collaborators in a group should be enabled. This means determining the lowest denomination of communication that is possible from a mobile device and providing additional functionality where possible. In this case, the most basic form of communication is by text, where a mobile application can take basic input from what is usually an alphanumeric keypad. By supporting text-based messaging as a first step, we can guarantee all participants can communicate at the very least this level. Mobile phone devices obviously support voice communication, although applications building on Mico will need to be able to utilize the voice hardware functionality. Many more feature rich mobile devices also support video recording and playback. The Mico Communication Service is designed to support all three modes of communication where possible.

Content: Facilitating communication between collaborators can be augmented with a content sharing service to allow collaborators to share data (which may provide topics for discussion) and to generate data that may be of interest to fulfilling the group tasks at hand. Sharing of data is essential in any collaboration, where sometimes communicating the data explicitly through group discussion may not be easy due to the amount of data or where data can not be easily described purely through communicative description. The Mico Content Service is designed to support search, retrieval, and publishing of arbitrary content.

Users: In any collaboration, we must be able to identify users in a meaningful way depending on the group's context. In a system to support short-term group collaborations, identities do not have to be persisted for long periods of time. Building on this assumption, Mico is designed to allow users to create their own arbitrary names that should be useful within a group's context. No restrictions on how names are chosen are enforced to allow groups as much freedom as possible to self-organize autonomously. A user is viewed as a person using a single device that is running a Mico service or application that is in-turn running a single instance of the JXTA platform.

It is important to understand that although a user is considered as a person using a single device it should not be construed that a user is tied to a single device running the JXTA platform. The JXTA specification detaches the concept of a user from a peer, citing the reason that this decoupling allows users to migrate from one peer to another.

Groups: To represent each organized collection of users, groups are utilized to manage and define partitions within the wider community (groups within groups) such as classes of students in a school, clubs within a university, or project teams within a company. Mico provides the mechanisms for allowing users to create and join groups, directly based on JXTA Peer Groups that provide scope and context for collaboration.

To support micro-collaborations, we make two assumptions from a user perspective:

- All communications amongst collaborators is of interest and of value to all other participants. There should not be a need for any private communication channels within the group, and Mico does not explicitly support one-to-one channels.
- All content shared amongst collaborators is of interest and of value to the whole group. Like with communications among participants, there should not be a need for private content within our group collaboration scenarios, and the system therefore does not explicitly support one-to-one content sharing.

The rationale for making the assumptions that require all participants to have ready access to all information generated by the group is two-fold. First, the collaborative scenarios and tasks that we aim to facilitate are assumed to be in relatively small groups (groups of less than around 10 participants), so implementing a replicated content repository is more easily realized since the size of the shared repository should remain manageable. This also provides a more responsive user experience in cases where connectivity is lost and devices need to re-establish connections to the network. Second, having a group communication system whereby all participants communicate over a single channel maps directly to how a face-to-face group would communicate. When a group needs to coordinate itself into subgroups or reorganize into different groups, Mico allows the freedom to create and manage multiple ad-hoc groups in situ.

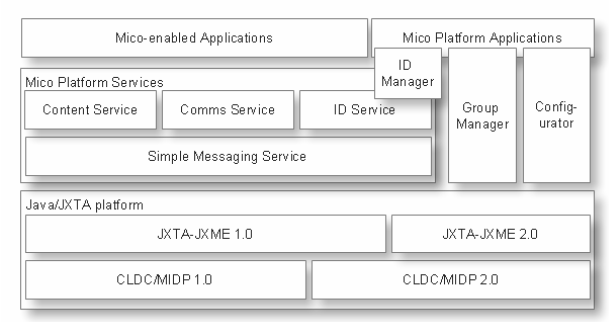

Figure 1. The Mico Platform Architecture.

Figure 1 illustrates how the Mico Platform software has been designed as a layered architecture, and is made up of the following:

- *Java/JXTA platform* - The Java ME reference implementations of the JXTA protocols, along with two of the Java ME platform configuration and profile combinations, make up the basis for the Mico platform implementation for the range of mobile devices that implement the CLDC/MIDP configuration and profile combination, and that the Mico project aims to address.
- *Mico Platform Services* - Composed of four basic services, this layer provides the network and logical functionality that mobile software developers can build on to create collaborative applications. The design includes services to enable content sharing, user-level group communication, and identity management. Each of these services is in turn built on top of a core generic messaging service that facilitates service-level group communication.
- *Mico Platform Applications* - A set of utility applications are bundled into the Mico platform at the application layer, where mobile software developers can include these applications into their own software projects to aid in the management of their own collaborative applications. This set of applications includes an identity manager, group manager, and a platform configuration application.
- *Mico-enabled Applications* - This part of the application layer consists of applications that build on the Mico Platform Services, where any applications that build on the Mico Platform Service layer are referred to as being "Mico-enabled". Mico-enabled applications can be bundled with Mico Platform applications into the same application suites, and conversely are not required to be bundled with the platform applications if they provide the functionality themselves.

C. Platform Services

The Mico Platform Service layer shown in figure 1 is made up of three basic services that application developers can build on: the *Communications Service*, the *Content Service*, and the *Identity Service*. The *Simple Messaging Service* serves as a core service that the other three build on. The intention of layering the services is to give software developers the ability to access any level of the service stack. Developers can therefore build on the core Simple Messaging Service to create their own services rather than being limited to building on the other three services. In this section we describe each of the Mico Platform services.

Simple Messaging Service: The Simple Messaging Service is the core service that all other Mico services are based on. Mico is designed to use a single shared communication channel, provided by JXTA, to communicate within a single group. To this length, the Simple Message Service provides the messaging capabilities for the higher-level services such as the Communications and Content Services that are discussed later. In order to correctly utilize the JXTA-based communication channel, the Mico Simple Messaging Service wraps JXTA messages as service specific messages for developers building collaborative applications, and provides a convenient sending mechanism and appropriate event notification when messages are received. The Simple Messaging Service acts as a base for all other service implementations, forming a universal messaging layer in the Mico service stack.

Communications Service: The Communications Service is designed to enable basic communication between users that belong to the same collaborative group. Building on the Simple Messaging Service, the Communications Service provides a service that enables users to send media messages to a group, and receive media messages from other participants of the group. Instead of providing a synchronous communications service, the Mico Communications Service is designed to enable basic forms of communication in short, fast, and asynchronous bursts. By building on the Simple Messaging Service, the Communications Service attempts to treat large messages (e.g. audio and video messages) in an atomic manner. Where an audio or video message starts sending over the shared channel, any other peers trying to send at the same time are not be able to do so. This allows all receiving devices to have maximal bandwidth available since they will not be sending data at the same time as receiving data, resulting in propagation of audio/video messages to the group being optimized across the underlying network infrastructure.

Identity Service: The underlying messaging systems in Mico rely on unique identification of each peer in a group. Therefore arbitrary user-defined identifiers cannot be used at the application level as there would be an increased likelihood of duplicate names being used within each individual Mico group, including the globally accessible Mico root group. There needs to be a mechanism for preserving the use of unique IDs for the purpose of message routing and propagation, and still allow the use of user-level IDs. According to Williams [6], IDs can be considered as a kind of awareness or presence mechanism in collaborative groups that allows participants can keep track of individual's actions. In Mico, peers are configured with an arbitrary ID that can be set to any `String` value, as determined by a user-level application that utilizes the Mico Identity Service. To distinguish individual users, the Identity Service provides a mechanism where users can select their own IDs mapped on top of JXTA-specific universally unique IDs (UUIDs) that are automatically assigned to JXTA peers. These user-level IDs are stored in a wholly replicated ID map, where each peer periodically broadcasts to the group their current user ID to ensure the ID maps are as current as possible.

Content Service: To allow mobile users to share and search for data in a Mico collaborative group, the platform provides the Mico Content Service. There are three main use-cases for a content service: Publishing content, searching for content, and retrieving content. Building on the Simple Messaging Service's paradigm of communicating all messages to all users, the Content Service makes use of the broadcast of all messages to keep a temporary cache of data not addressed to a peer, with the assumption that by disseminating information on what each user is searching for or retrieving, the group context will more likely require access to the data being cached. This may be useful since it keeps the cache in a state where the most recently accessed or requested data is made available in the caches throughout the peer group, and therefore makes the data of most recent interest (to someone in the group) the most readily available to the rest of the group. To assist in the search mechanism where the content service is entirely decentralized, we use a replicated content metadata index so that only content retrieval messages make up the bulk of content service message broadcasts.

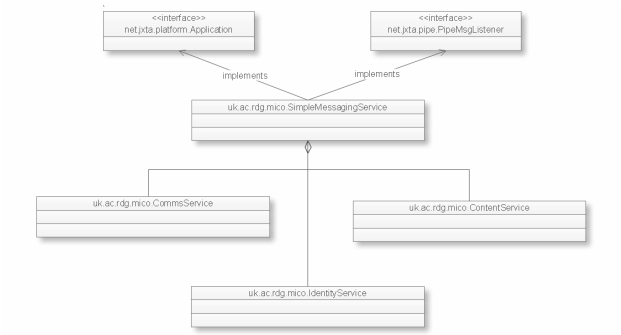

Figure 2. The relationships between the Mico Platform Services.

The class diagram shown in figure 2 illustrates the relationships within the Mico Platform service set. Each of the more application specific functionalities exposed by the Communications, Content, and Identity services directly utilizes an instance of the Simple Messaging Service. The result is that all of the higher level services communicate through a single common messaging service. This reduces the underlying software complexity in terms of the number of threads needed to maintain network connections, and in the amount of memory needed to maintain multiple service instances. Only the Simple Messaging Service implements the JXTA specific networking functionality used to maintain group communication.

D. Platform Applications

In addition to the service layer provided by Mico, the platform provides three application level tools to facilitate configuration of the network settings, creation and management of groups, and managing identities. Each of these applications is implemented as a graphical Java ME application that can be packaged into a developer's software project by bundling them into deployable application suites, and also provides functionality packaged in the Mico library to allow Mico-enabled applications to hook into the tasks normally carried out by these applications. The Platform Applications are described as follows.

Configurator: Although packaged applications may provide some default settings, possibly hard-coded, a general platform configurator application is provided as one of the Mico Platform Applications. This application allows configuration of the Mico platform, including setting up an access point to the underlying JXTA network backbone, transport protocol preferences, and local cache settings and policies. Normally these settings are hidden from the user and defaults provided by the application developer or deploying administrator.

Group Manager: Every Mico user needs the capability to manage and manipulate collaborative groups. The Group Manager application is a tool for carrying out these tasks, and builds directly on the JXTA platform. The Group Manager application provides a graphical interface to create groups, find and join groups, and resign/leave groups. Mico maps directly on top of JXTA Peer Groups, and therefore the Group Manager is implemented to interact directly with JXTA to maintain Mico groups.

Identity Manager: The Identity Manager application allows users to select a human-readable identifier for identifying themselves to other users in a Mico group. Users can choose their own IDs using the Identity Manager, which in-turn propagates the mappings using the Identity Service. Mico users may have multiple IDs which can be different in every Mico group that they are a member of.

III. EXEMPLARY APPLICATIONS

To demonstrate using the Mico Service library in real applications, a set of exemplary applications were developed. These applications include a multimedia chat messenger (MultiChat), a content sharing application (Content Safari), and shared mobile web browser (Shared Microbrowser). MultiChat and Content Safari's features build directly on Mico's basic collaborative services, namely the Communications Service, Content Service and Identity Service, whilst the Shared Microbrowser application builds on the Simple Messaging Service to show an example of extending the Mico Platform with a custom built service. Although there are many other systems that provide similar functionality to these examples, the applications described in the following subsections aim to show how developers can easily build similar collaborative applications with the Mico Platform library using relatively few lines of code. The code examples shown in the following subsections should only be considered for illustrative purposes and should not be considered as a complete instruction on the Mico library and software's usage.

A. Mico MultiChat

To show how the Communications Service can be used to enable group communications, we created a multimedia messaging application, *Mico MultiChat.* MultiChat does not provide a real-time synchronous conferencing client; one must remember that the Communications Service only supports asynchronous messaging where the aim is to provide a fast multimedia messaging system without implementing real-time communications.

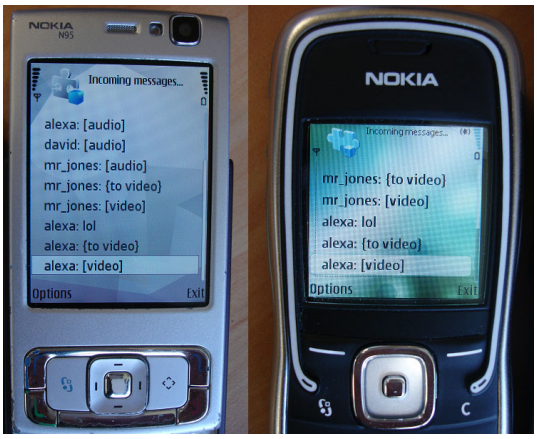

Figure 3. MultiChat: A group communications session running on a Nokia N95 and 5500 Sport respectively.

Users can switch between the available media messaging methods, where a consensus can be reached within individual groups to which messaging modes are most appropriate. By allowing groups to self-organize, we aim to cater for unpredictable configurations of groups – providing a messenger that is audio only or video only may hinder progress if there are collaborators who are not able to interoperate within those static configurations. Apart from the technical configuration, we also aim to cater for user preferences by providing the range of options. On receiving a message, a notification is added to a communications feed (figure 3). Text messages are displayed directly in an on-screen feed. Notifications of receipt of audio and video, along with who sent the messages, are also shown. Where a media message needs to be played back, a user can just select the entry in the feed to start playback. The underlying data is discarded after a single playback in order to keep memory usage a low as possible.

The Communications Service is utilized by MultiChat by getting an instance of the `CommsService` class with an appropriate `CommsServiceEventListener` implementation that hooks back into the application specific code. The `CommsService` class provides a single method, `send(CommsMessage m)`, that takes any instance of a valid `CommsMessage`. The different subclasses of `CommsMessage` include the following different media messages: `TextMessage`, `AudioMessage`, and `VideoMessage`. Having a uniform interface for implementing different kinds of multimedia communications hides much of the complexity that underpins group messaging at a network level. By implementing an event listener interface, notification of receiving messages can also be dealt with at the application level. The following code snippet illustrates how to send and receive messages in only a few lines of code.

```
Platform p = Platform.getInstance();
CommsService comms = (CommsService)p
   .getService("uk.ac.rdg.mico.CommsService");
comms.addListener(new CommsEventListener() {
      public void received(CommsMessage m)
      {
         if (m instanceof TextMessage) {
            // hook back into application here
         }
      }
   });
TextMessage t = new TextMessage("Good morrow!");
comms.send(txt);
```

In the code example, we create an instance of the Mico platform which has already dealt with the underlying JXTA initialization and connectivity to the backbone network. We get an instance of the `CommService`, supplying it with an event listener. The event listener interface must be implemented here and is done so with an anonymous class. In this case whenever a message is received by the `CommsService`, the `commsEvent` method is executed. How the method is implemented is entirely up to the developer, and in this example, we check the class type to filter out text only messages. The last two lines illustrate how to send a message by creating a `TextMessage` object and sending it with the Communication Service simply by providing the object to the send method.

B. Mico Content Safari

To demonstrate the use of the Content Service, we developed a mobile content sharing application called *Mico Content Safari*. This application lets users publish any kind of files to a collaborative group, including multimedia generated on the device with other applications. Content Safari accesses the underlying phone file system directly, and therefore does not need to build into the Mico-enabled application the multimedia gathering functionality such as audio recording, image capture, and video recording. We can rely on a phone being capable of capturing these kind of media based on the onboard hardware already being accessed by native pre-installed applications.

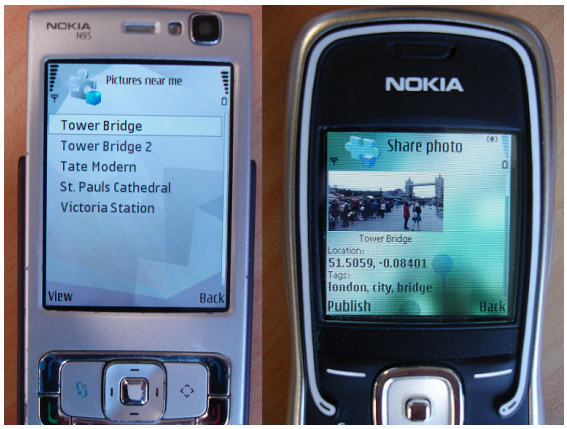

Figure 4. Content Safari: Searching for content near "my location" on a Nokia N95, and publishing a photo on a Nokia 5500 Sport.

Content Safari also integrates with the Java ME Location API which allows us to determine an estimate of the current location when the application is running. Within the application we can add location as metadata to the files being shared. For example, and as illustrated in figure 4, a user might want to publish a photo of a famous landmark such as Tower Bridge in London. Another user in the group may want to find Tower Bridge but does not have any idea what it looks like or how far away it is. Content Safari can order search results by distance from a user's location, organizing them according to location-based contexts.

The Content Service is utilized by Content Safari by getting an instance of the `ContentService` class with a `ContentServiceEventListener` implementation that

hooks back into the application specific code, in the same way that was shown in the earlier `CommsService` example. The `ContentService` class provides two methods to send search queries and to request to retrieve content items. The following example shows how to instantiate the Content Service.

```
Platform p = Platform.getInstance();
ContentService content = (ContentService)p
   .getService("uk.ac.rdg.mico.ContentService");
content.addListener(new ContentEventListener() {
      public void queryResult(QueryResult r)
      {
         // do something with the result
      }
      public void transferEvent(TransferEvent t)
      {
         // do something when the
         // transfer completes
      }
   });
```

The Content Service's event listener notifies with two events. The `queryResult(QueryResult r)` event notifies of results retrieved after making search queries. The `QueryResult` objects contain information relating to specific content including metadata and a unique identifier which is used to retrieve content. The `transferEvent(TransferEvent t)` method notifies about different events that may occur during a content transfer. Transfer events might include notifying of completed transfers, erroneous events, or reports on the progress of a transfer.

To publish content, the `publish(String uri, HashMap meta)` method adds the resource pointed to by a URI to the local user's share list. The key-value pairs found in the `meta` parameter map directly to application-defined metadata.

```
String loc = "file:///TowerBridge001.jpg";
Map meta = new HashMap();
meta.put("mime", "image/jpeg");
QualifiedCoordinates qc = locationProvider
   .getQualifiedCoordinates();
meta.put("longitude", qc.getLongitude());
meta.put("latitude", qc.getLatitude());
meta.put("longAccuracy", qc.getVerticalAccuracy());
meta.put("latiAccuracy", qc.getHorizontalAccuracy());
content.publish(loc, meta);
content.get(queryResult.getID());
```

Having a generic metadata container allows application developers to put any kind of metadata with the published content, where the Content Service deals with serializing and decoding the maps. In the example above, we add some fields to publish the location data of a content item, here we get the location data from the a Java ME `LocationProvider` instance, and also include a field to describe the content's Internet MIME type, in this case being a JPEG image. The final line above illustrates how to request a content item by using a unique ID retrieved from a query result using the `get(String contentID)` method.

C. Mico Shared Microbrowser

The *Shared Microbrowser* application is a modified version of the open-source *JCellBrowser* mobile application. The JCellBrowser project [7] is a Compact HTML (C-HTML) browser that is designed to be lightweight and portable across devices implementing the Java ME CLDC/MIDP platform. C-HTML is a subset of HTML that was widely adopted by i-mode mobile phone devices and described in detail by Kamada in [8]. As a proof of concept, we took the source code from JCellBrowser and created a new service based on top of the Mico Simple Messaging Service to allow collaborative sessions where users can share a single view on through a common application. Like some desktop shared Web browsers, the approach taken is to propagate a simple event from a user that indicates the Web URL that the group should all be viewing, and rendering the Web page using the standard local browser code.

This was achieved by defining a new protocol message that simply propagates any change in a user's URL to the other collaborators, and using the Simple Messaging Service to propagate the messages. A custom message listener is created to filter out these URL update messages from the Simple Messaging Service, which in turn is passed back to the Shared Microbrowser application via a custom listener attached to our new service.

The following code snippet shows how we implement a new Mico service. Before creating our new service class, we must define an event listener interface for the service. In this case we define a `UrlEventListener` that is implemented to pass the URL updates to the application level.

```
public interface UrlEventListener {
   public void urlUpdateEvent(String newUrl);
}
```

Next we also define new protocol message, in this case a single protocol message, `UrlUpdateMessage` that extends the `SimpleMessage` class to ensure compatibility with the underlying messaging service. The message is a one-way propagation to the group, so no reply messages need to be defined.

```
class UrlUpdateMessage extends SimpleMessage {

   private String url;

   public UrlUpdateMessage(String sender,
      String messageID, String url) {
      super(sender, messageID);
      this.url = url;
   }

   public UrlUpdateMessage(Message message) {
      super(message);
      this.url = message
         .getMessageElement("propUrl", "url")
         .toString();
   }

   public Message toJxtaMessage() {
      Message message = super.toJxtaMessage();
      StringMessageElement urlElement =
         new StringMessageElement("url",
         new String(this.url));
      message.addMessageElement("propUrl", urlElement);
   }

   public String getUrl() {
      return this.url;
   }

}
```

The `UrlUpdateMessage` implementation adds to the basic functionality provided by the `SimpleMessage` class. `SimpleMessage` objects provide basic message information such as a unique message ID, the sender's

unique ID, and a timestamp. `UrlUpdateMessage` adds to those fields to propagate a URL change.

Finally we can define our new service, `PropagateUrlService`. Here we define any public interfaces for the application developers to use, in this case with the `urlChanged` method providing the functionality of propagating URL changes. The bulk of the logic occurs in the constructor which takes the reference to our event listener defined earlier, and links it into the Simple Messaging Service. We retrieve an instance of the Simple Messaging Service, create and register a `SimpleMessagingEventListener` that we use to in-turn notify the application level via our service specific listener.

```
public class PropageteUrlService {

   SimpleMessagingService s;
   UrlEventListener listener;

   public PropagateUrlService(UrlEventListener l) {

      this.listener = l;

      s = (SimpleMessagingService)Platform
         .getService("uk.ac.rdg.mico
            .SimpleMessagingService");

      s.add(new SimpleMessagingEventListener() {
         public void msgEvent(SimpleMessage m) {
            if (m instanceof UrlUpdateMessage) {
               (UrlUpdateMessage)m;
               listener.urlUpdateEvent(urlUpdate
                  .getUrl());
            }
         }

      // public method for application to use
      public void urlChanged(String url) {
        UrlUpdateMessage u = new UrlUpdateMessage(url);
        s.send(u);
      }
   });
}
```

Modifying the JCellBrowser code to integrate this new Mico-enabled service only takes a few lines of code. The following code snippet shows how the new service is instantiated, and a custom listener is added to integrate with the microbrowser at the application level. In this case whenever an event is received, it calls a method within the Web browser application that refreshes the page being viewed.

```
PropagateUrlService propUrl = new
PropagateUrlService(new PropagateUrlEventListener() {
   public void urlUpdateEvent(String newUrl) {
      WebBrowser.this.setPage(newUrl)
   }
});
```

Finally, to send an update event, we use the method defined in the original service class to propagate the URL changes, as shown in the following example.

```
propUrl.urlChanged("http://acet.reading.ac.uk/");
```

Note that for the sake of clarity, we have omitted much of the application specific code in order to illustrate how the Mico Platform library was used in these exemplary applications. For this example, we only propagate the event of changing what URL is being browsed, where the result is that all users should be viewing the same page, and, in this basic case, all users have "control". If more complexity is to be built into a shared Web browser service, a possible extension could be to propagate user input events while browsing in order to share item focus with a group. This would be in a similar vein to having a telepointer shared between desktop applications as implemented in some groupware systems.

IV. RELATED WORK

The work described in this paper is a direct evolution of *MicroCoco,* part of the *Coco* project that developed a P2P platform for ad-hoc group formation and collaboration and described in [9]-[11]. Coco was developed as a P2P desktop platform for software developers to build Java groupware applications, and MicroCoco aimed to interoperate with the full desktop version to provide collaboration services across the device domain. However due to limitations of the underlying platform support for JXTA for mobile devices at the time, pervasive connectivity between Coco and MicroCoco was possible but very inefficient. There was also a significant added complexity to bridging between Coco and MicroCoco as many of the collaboration services could not be built on the same underlying JXTA service set. As a result, Mico was developed with a view to building upwards from the lowest common denominators (i.e. micro-collaboration services) with a design ethos based around reducing complexity in communication and processing.

A project based on similar technologies, but with a different approach to enabling collaboration was carried out by the *ProMoCoTo* project. As described by Wang and Sørensen in [12], ProMoCoTo's aim was to promote spontaneous collaboration using a P2P mobile application based on Java technologies. Their approach, like Mico, was to build on Java and P2P technologies. Although also having assessed JXTA as a networking solution, the authors opted to use a different P2P framework, *Proem*, developed by Kortuem et al [13] at the Wearable Computing Laboratory, University of Oregon. Proem aims to provide a framework for rapid development of Java applications aimed at ad-hoc mobile network environments, and like JXTA is designed to be independent of the underlying transports. Wang and Sørensen justify their choice of underlying networking platform by reasoning that Proem aims to be more resilient in unpredictable network environments and focuses on user-to-user communication rather than dealing with lower-level elements. However our own design decision of choosing a JXTA-based approach was based on the assumption that most mobile users will actually have a relatively stable Internet connection (either Wi-Fi or 2G/3G) rather than having to form ad-hoc mesh networks. Also, Proem is based on Java SE making it inherently less portable in the mobile device domain than Java ME solutions. The aims of ProMoCoTo were also vastly different from that of Mico. ProMoCoTo was a specific tool aimed to automatically connect and communicate with peers in its network neighbourhood without user intervention, with a view to suggesting to users that real-world collaborations can take place. Mico aims to be a generic collaboration platform, and one which tools can be built on.

V. CONCLUSION

This paper describes our efforts to develop an easy to use software platform for application developers to create micro-collaborative applications in a diverse mobile device environment. We describe a set of services built on Java and JXTA technologies that can be used by application developers to create a range of mobile applications, and demonstrate their use through a set of example applications. These examples illustrate the ease of use of the software library, and additionally show how by exposing the lower level platform service of the Simple Messaging Service, developers can extend the platform to create custom services for applications that do not fall into the pure communications/messaging or content sharing categories. Having built on JXTA as our the P2P messaging enabler, in the future we aim to build towards cross-device category bindings of Mico based on Java SE and its corresponding JXTA build. We also plan to develop domain specific applications on top of the Mico platform, such as for the corporate work and educational settings, and to assess the performance and behaviours of the Mico platform itself in real-world deployments.

REFERENCES

[1] M. Wiberg and Å. Grönlund, "Exploring Mobile CSCW: Five Areas of Questions for Further Research," in Proc. of the 23rd Conf. on Information Systems Research, 2000 © Univ. Trollhättan, Uddevalla, Sweden.

[2] S. Kristoffersen and F. Ljungberg, "Making Place to Make IT Work: Empirical Explorations of HCI for Mobile CSCW," in Proc. ACM SIGGROUP Conf. on Supporting Group Work, Phoenix, AZ, 1999, pp. 276-285.

[3] J. Brandt, N. Weiss and S.R. Klemmer, "Designing for Limited Attention," in Stanford Univ. Computer Science Technical Reports, 2007 [Online]. Available: http://hci.stanford.edu/cstr/

[4] J. Roth, "Seven Challenges for Developers of Mobile Groupware," in Proc. ACM SIGCHI 2002 Conf. on Human Factors in Computing Systems, Minneapolis, MO, 2002 [Online]. Available: http://www.wireless-earth.de/paper/chi02ws.pdf

[5] L. Gong, "JXTA: a network programming environment", IEEE Internet Comput., vol. 5, no. 3, pp. 88-, May 2001.

[6] J.P. Williams, "Groupware: Shared Thoughts, Shared Media, and Shared Models," School of Information at the Univ. Texas at Austin, 2003. [Online] Available: http://www.ischool.utexas.edu/~i385tkms/blog/archives/patrick/groupwarepaper.html

[7] JCellBrowser: A J2ME cHTML Browser. [Online] Available: http://www.sourceforge.net/jcellbrowser

[8] T. Kamada (1998, Feb. 9), *W3C Note: Compact HTML for Small Information Appliances.* [Online] Available: http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/

[9] I.M. Bhana and D. Johnson, "Supporting Ad-hoc Collaborations in Peer-to-Peer Networks," in Proc. ISCA 17th Conf. on Parallel and Distributed Computing Systems, San Francisco, CA, 2004, pp. 491-496.

[10] I.M. Bhana and D. Johnson, "Developing Collaborative Social Software," in Springer LNCS, Computational Science – ICCS 2006 Part II, pp. 581-586, May 2006.

[11] D. Johnson and I.M. Bhana, "Pervading Collaborative Learning with Mobile Devices," in Proc. Interactive Computer-aided Blended Learning Conference, 2007 © Kassel University Press

[12] A. Wang and C. Sørensen, "Mobile Peer-to-Peer Technology used to Promote Spontaneous Collaboration," in Proc. 2005 Symposium on Collaborative Technologies and Systems, St. Louis, MO, 2005, pp. 48-55

[13] G. Kortuem, "Proem: A Middleware Platform for Mobile Peer-to-Peer Computing," ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6, no. 4, pp. 62-64, Oct. 2002 (doi:10.1145/643550.643557)

AUTHOR

David Johnson was with the School of Systems Engineering, University of Reading. He is now with the School of Biological Sciences, University of Reading, Whiteknights, Reading, RG6 6BX, United Kingdom (e-mail: d.johnson@reading.ac.uk).

This work was supported in part by the Centre for Advanced Computing and Emerging Technologies (ACET Centre), a centre of excellence for computational sciences, based in the School of Systems Engineering at the University of Reading, UK.

Submitted 12 August 2009. Published as resubmitted by the authors on 15 September 2009.