2015 IEEE 14th International Symposium on Network Computing and Applications

@migo: A Comprehensive Middleware Solution for Participatory Sensing Applications

Rafael Bachiller, Nelson Matthys, Javier del Cid, Wouter Joosen, Danny Hughes
iMinds-DistriNet, KU Leuven
B-3001 Heverlee, Belgium
{firstname.lastname}@cs.kuleuven.be

Kristof Van Laerhoven
Faculty of Engineering
University of Freiburg, Germany
kristof@ese.uni-freiburg.de

***Abstract*—In the participatory sensing model, humans may serve as opportunistic sensors and flexible actuators while also consuming sensing services. Integrating humans into sensing systems has the potential to increase scale and reduce costs. However, contemporary participatory sensing software provides poor consideration of user dynamism, which includes: mobility across networks, mobility across devices and context-awareness. To address these limitations we propose the *User Component* and *User Bindings*. The former represents the user as a first class reconfigurable element of evolving and shared participatory sensing platforms. The latter allows the middleware to support multiple communications channels including Online Social Networks (OSN) to connect users with sensing applications. Our approach increases user participation, reduces out-of-context interactions and only consumes a limited amount of energy by sharing context information between applications. We support these claims by evaluating our approach on a two weeks experiment in which three participants take part in three concurrent participatory applications.**

Keywords*—*Participatory Sensing, Online Social Networks, Component-Based Software Engineering.

I. Introduction

Previous research [1], [2], [3] has illustrated the benefits of participatory sensing [4] where the incorporation of human participants [5], [6] with sensing systems is demonstrated to extend the scope and scale of sensing infrastructure. In the participatory sensing model, users contribute by providing and managing smartphone platforms that host sensing software. Furthermore, users may report observations that are difficult to measure with hardware sensors [7] or generate content that adds value to sensing scenarios, for example by validating sensor readings [6]. However, a key problem when including humans in sensing scenarios is managing the dynamism that humans introduce, which includes: mobility across networks, mobility across devices and changing operational contexts.

Contemporary participatory sensing applications [6], [3], [5] are typically built on a case-by-case basis. Software is developed to achieve a single static application goal for a specific hardware platform and with support for a fixed set of context assumptions. This monolithic approach to developing participatory sensing applications incurs high development effort due to a lack of support for software reuse. Furthermore, interactions between the user and applications are implicitly hard-coded into the application, making the resulting applications difficult to understand, modify or extend at runtime.

Recently, the research community has proposed several frameworks for developing participatory sensing applications [1], [2], however, these frameworks assume a static application goal and provide limited consideration of user dynamism. Most importantly, these approaches give little consideration to explicitly modelling interactions between users and the sensing application. We argue that participatory sensing applications should be built from assemblies of reusable software components which can be reconfigured over time to support evolving application requirements and changing user contexts. User-application interactions must be explicitly modelled in order to be able to understand, modify and extend these interactions.

In this paper, we present *@migo* (pronounced AT-me-go), a novel middleware solution to model and represent users in participatory sensing applications. @migo provides two key elements, the User Component and User Bindings. The *User Component* allows users to be modelled as a first-class software entities and to be dynamically reconfigured at runtime, for example by connecting users to new applications or disconnecting them based upon their current context. This allows users to be managed using the same well known tools used in Component-Based Software Engineering (CBSE) [8].

The *User Binding* is a complementary interaction mechanism that provides communication support between applications and users that exhibit mobility across both networks and devices. This is achieved by leveraging on the widespread availability of Online Social Networks (OSNs), such as Twitter or Facebook, which provide an alternate communication channel to reach the user when no application software is available or direct connectivity is restricted.

We implemented a prototype of the @migo middleware for Android [9] and the Twitter [10] social network. We evaluate the proposed solution via three participatory sensing applications running in a smart office sensor network: thermal comfort, heater monitoring and window state reporting. Our evaluation demonstrates that @migo can be used to: (1) reduce out-of-context interactions, (2) increase user availability and (3) only consumes a limited amount of energy by sharing context information between applications.

The remainder of this paper is structured as follows. Section II provides an summary of participatory sensing frameworks focusing on user's support. Section III shows our model of the user within participatory sensing applications. Section IV details our proposal for supporting mobility between networks and devices. Section V illustrates the proposed architecture, focusing on the case-study applications.

978-1-5090-1849-9/15 $31.00 © 2015 IEEE
DOI 10.1109/NCA.2015.26

IEEE computer society

Section VI evaluates the built prototype by showing the results obtained from real applications. Finally, Section VII concludes and discusses promising directions for future work.

II. Related work

Burke et al. [11] clearly state the classes of human interactions participatory sensing applications should implement, and identify the necessity of defining application-level mechanisms for supporting participatory sensing applications, an argument that is supported by Payton et al. [12]. In II-A we review frameworks for building participatory sensing applications. In II-B we review key participatory sensing applications. Finally, in II-C we enumerate the middleware requirements for supporting participatory sensing applications.

A. Participatory Sensing Frameworks

Several approaches for including users in distributed applications have been proposed in the literature. *Ohmage* [13], by Tangmunarunki et al., offers a generic, modular and extensible platform for participatory sensing applications including data from both users and mobile phones' sensors. The combination of smartphones with web-based approaches, context-aware interactions and new apps deployment at runtime are some design decisions also present in our approach.

ParticpAct [14], by Cardone et al., proposes a mobile crowdsending platform supporting multiple simultaneous *campaigns* performing active sensing actions (i.e. involving actively the user) and passive sensing actions (i.e. from mobile phone's sensors). The latter can be directly reported to the backend or can be used for inferring a user's context (e.g. activity recognition) in order to support user profiling, as we also do. Evaluation shows that ParticipAct reduces the number of out-of-context interactions as well as the number of users that are required to be involved in a task.

PRISM [15], by Das et al., proposes a system balancing generality, security and scalability. Its *push model* enables the server to deliver a task to the group of nodes matching the requirements. The server is periodically updated with context information from the devices. Our approach also allows the definition of user groups based on context, however, in contrast to PRISM, we also allow mobile devices to autonomously select which group they should join to.

Table I compares these three representative systems with our proposal. Although all the platforms support contributions from both users and sensors, only @migo offers a fine grained identification schema supporting multiple devices belonging the same user. Regarding with the set of interactions defined to the user, Ohmage and @migo implements a full and independent set meanwhile ParticipAct and PRISM only offer a reduce set. Focusing on the triggering mechanisms for the interactions, all platforms support time and context based triggering. Additionally, Ohmage and PRISM offer context-only triggered interactions due to their expressive language for defining participatory sensing applications. Although all frameworks support communication through an application, only Ohmage and @migo also offer a web platform for interacting with the user. To the best of our knowledge, @migo is the only platform including OSNs. Furthermore, @migo also supports mobility between devices. As summary, we find prior systems provide poor support for managing user dynamism as they provide little support for mobility between devices or multiple communication channels as OSNs.

TABLE I. Comparison between representative participatory sensing platforms and @migo

	Ohmage	*PartcipAct*	*PRISM*	*@migo*
Sensors' and users' contributions	✓	✓	✓	✓
Multi device support				✓
Generic set of users' interactions	Full	Partial	Partial	Full
Time and context based interactions	✓	✓	✓	✓
Communication channels	App and Web	App	App	App, Web and OSNs
Users' mobility between devices				✓

Podnar et al. [16] illustrate the benefits of using a publish/subscribe middleware for supporting mobile crowdsensing applications. In addition, they also explore several mechanisms for supporting mobility between networks, including a REST approach (but not for all kind of connections) and platform specific protocols (e.g. Google Cloud Messaging). Our proposal offers similar features, but also includes mobility between devices and support of multiple communication channels.

B. Participatory Sensing Applications

Balaji et al. [17] define a participatory sensing based solution for bridging the gap between metering, monitoring and control within a smart building context. They implemented a web application for obtaining feedback from the users, offering energy saving suggestions and modifying configuration settings in the heating ventilation and air conditioning (HVAC) system. As this paper is application-driven, it does not explicitly define how a user can interact with the system, or how to group users based on their context. @migo supports a superset of the interactions necessary to realise this application while providing more advanced support for user's dynamism.

Hicks et al. [18] propose a system called AndWellness for collecting data from users and phone sensors within a healthcare context. Although they define a platform for supporting multiple *campaigns* (i.e. participating in a experiment), user's roles cannot change at runtime. Therefore, the organiser cannot manually form a sub group of participants based on their role. The concept of *triggers* allows delivery surveys at realtime, however, this represents a subset of the interactions supported by @migo. Finally, no specific support for user mobility is provided.

Paxton et al. [6] perform two real word experiments that include both values provided by sensors and human contributions within an environmental monitoring context. Although they discuss the implications of including humans, no model is proposed for supporting human interactions with a participatory sensing application neither no support for users' roles. Furthermore, each experiment requires different applications running in the mobile device without any software reusability.

Demirbas et al. [5] propose Twitter as an open platform for running participatory sensing and crowdsensing applications. Their approach is validated through a participatory weather monitoring application also within environmental monitoring context. Based on real-world experiments, they model not only the response time of the user but the quality of the provided data. Nevertheless, no model for human participation is provided as they define a specific syntax for taking part in the participatory sensing application, and no support is provided for gathering data from mobile phone sensors neither supporting different user roles. While using Twitter as a communication channel handles user mobility between devices, this approach provides no support for the execution of local applications reducing its scope of applicability.

C. Middleware support for participatory sensing applications

Based on the frameworks and applications previously considered and on the analysis performed in [19], we first identify the following set of middleware requirements to support the creation and execution of participatory sensing applications. We then motivate why component-based middleware addresses these requirements.

Heterogeneity. Users within participatory sensing applications handle devices with different capabilities, features and operating systems.

Interoperability is required between different mobile devices, regular computers and powerful backend devices.

Reconfiguration support is demanded for adapting the allowed user's interactions according to his context.

Distributed Relationships are inherently needed to support participatory sensing applications and, joint with local relationships support, enhances component reusability.

Energy Efficiency and Performance. Battery powered devices handled by the user run multiple applications in addition to the middleware, so a conservative behaviour is expected in terms of battery and CPU consumption.

Component-based middleware provides a good fit to address these challenges, as it introduces software components as common abstraction to handle heterogeneity, while still allowing the strengths of each platform to be fully exploited. The use of common networking and data exchange standards provides interoperability for facilitating components running on heterogeneous devices into coherent applications. The midlware also supports reconfiguration of software functionality and allows efficient reification of the current state (i.e. discovery and analysis), including flexible binding modalities, distributed relationships and a clean separation between local and distributed functionality. Lastly, it is primarily designed to consume minimal energy and offer good performance (even on mobile devices as smartphones) while imposing minimal burden on the component and application developer.

III. The User Component: a Software Abstraction for Modelling Users

The goal of the User Component is to model and represent a user in a participatory sensing application. At the core of our approach is the conviction that the user should be represented as a first-class software component and, as such, developers should be able to reconfigure and inspect a user using the same tools that are used to deal with software entities. As shown in Figure 1, the User Component provides four software interfaces: interfaces on the right (*Request* and *Notification*) represent communication from the participatory sensing system to the user and interfaces on the left (*Reply* and *Unsolicited Input*) represent communication from the user to the participatory sensing system. In the section, the model will be analysed from the application plane (interactions) and management plane (lifecycle) perspective.

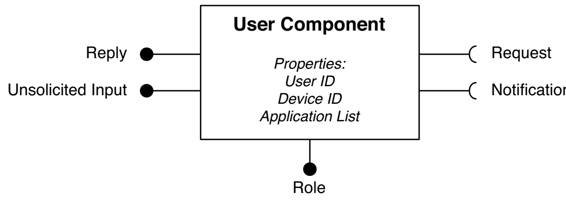


Fig. 1. The User Component provides four software interfaces (right and left) allowing interaction from/to the participatory sensing system. The lifecycle of the user in each participatory sensing system is supported by both the properties and one software interface (bottom).

A. Supported interactions

The User Component defines two required interfaces and two provided interfaces. All of these interfaces are unidirectional, providing the fundamental mechanisms for communication between the user and the sensing application. The interfaces may be accessed locally or remotely, they are:

- *Request.* This required interface provides a mechanism to request input from a user.
- *Reply.* This provided interface provides a mechanism for users to respond to queries received on the 'Request' interface.
- *Unsolicited Input.* This provided interface allows users to push unsolicited information to a sensing system (e.g. reporting that an traffic accident has occurred).
- *Notification.* This required interface enables sensing systems to push unsolicited information to users without requiring a reply from them (e.g. prompting the user to close a window).

The interaction mechanisms provided by the User Component map well with those proposed by Burke et al. [11] who argue that the data gathering process in participatory sensing applications can be triggered by the network (modelled by the interfaces *Request* and *Reply*), by the user (modelled by the interface *Unsolicited Input*), or is continuously running to obtain data from mobile phone sensors (in which case the interaction is realised by specific components running on the device itself). In addition, the *Notification* interface allows the sensing system to push information to its users, for example, to close the feedback loop and interact with the user after he provided a value to the application.

The User Component also supports an extensible set of properties, each of which is a tuple composed of a key and a value. Properties may be accessed remotely in a request/reply fashion. The User Component defines a number of well-known properties that: (i.) identify a user and link that user to (ii.) their current device and (iii.) installed applications. This allows participatory sensing systems to reason over the current context of their users and, where necessary, enact change through the deployment of full applications, or individual software components or the reconfiguration of communication abstractions.

B. *Supporting the software lifecycle*

Users decide about which applications to participate in and when start and stop participating in each one. As each participatory sensing application may require not only the User Component but also other auxiliary elements, users rely on the User Component to manage the process of consistently installing, updating or removing specific software pieces for each participatory sensing application.

The *Application List* property allows to control the set of applications a user is participating in. It is an array containing the name of the participatory sensing applications that are currently running on the device. Each element of the array also contains a reference to all software components running on that device that compose a particular participatory application.

Applications may be remotely installed, updated or modified by updating the Application List property, which causes the supporting middleware to deploy, instantiate and configure the required components.

C. *Supporting user dynamism*

We define *user's role* as the available set of interactions of the user within each participatory sensing application. For example, only users with role 'running in the gym' (based on the GPS and output from activity recognition algorithm) are allowed to contribute with an Unsolicited Input. The array *Application List* contains the active user's role for each installed application. So, it allows any other component (e.g. GPS or activity recognition) to inspect or modify the role by only updating this property. Any change in the user's role requires reconfiguring the user inside the application. The application-specific component called *Reconfiguration Entity* performs those (re)configuration as requested by the User Component through the interface *Role*. This approach makes the user role's detection independent of the User Component maintaining a generic, modular and extensible design.

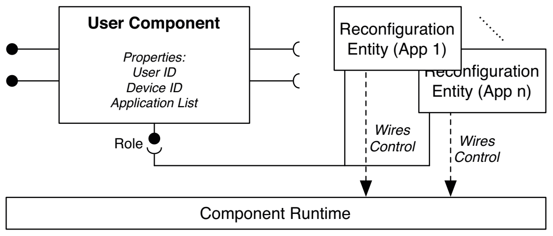


Fig. 2. The Reconfiguration Element prompts the middleware to perform the required reconfiguration actions according to the user's role.

IV. User Bindings: an Interaction Model for Extreme Mobility

It is increasingly common for users to use several devices (e.g. computer, smartphones and tablet) in a sequential fashion. The capabilities of those devices are heterogeneous in terms of connectivity and available software.

The User Bindings provide a consistent communication abstraction to the middleware that maps to multiple communication channels in order to support mobility between networks and devices. Our previous research [20], [21] revealed that using Online Social Networks (OSNs) as higher level communication channels significantly improves the availability of the users in participatory sensing scenarios. We apply this approach in the design of user bindings.

A. *Identification schema*

Network layer identifiers (e.g. IP address) are widely used for representing non-mobile devices. Although several protocols at the network layer have been defined for supporting mobility between networks (e.g. Mobile IP), they are not yet widely deployed. An application layer mechanism is therefore required which allows for communication in the face of changing network layer addresses. These mechanisms usually rely on an application specific *device identifier*. Additionally, state of the art applications usually allow the user (represented by an *user identifier*) to be logged in multiple devices simultaneously. Although the user only actively uses one device at each moment, these applications don't offer proper mechanisms to communicate selectively with that device. Hence, mobility between networks and devices requires an identification schema to decouple the user identifier from the device identifier while also separating the device identifier from its actual network address.

We propose an inverted-tree identification schema (Figure 3) allowing the participatory sensing system to either selectively communicate with the device the user is currently operating, or to communicate with a user's device without regarding its network address. The root level is composed by globally unique *user identifiers* (e.g. Twitter account identifier). The following level is composed by *device identifiers*, unique within user's domain (e.g. smartphone). The last two levels, *component identifier* and *interface identifier* are specific for software entities and the services they provide.

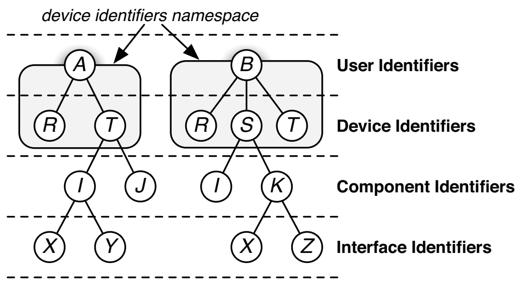


Fig. 3. The identification schema relies on levels for adapting the granularity of the identifier according to the element you want to communicate with.

B. *Mobility across networks*

Every time a device changes network, its network address may also change. Furthermore, the device itself may not be aware of its public IP address in cases where it is behind a NAT box. As demonstrated by other protocols such as

STUN [22], an external service on a public IP address is required to mediate communication between devices behind a NAT. Nevertheless, applying this approach to mobile and battery-constrained devices (e.g. smartphones) reduces lifetime and increases traffic as each application connects with its own external service. So, state of the art operating systems offer a platform specific, common for the applications and well-integrated messaging cloud (e.g. Google Cloud Messaging for Android) for connecting from a public external server to the mobile device.

In order to support mobility between networks without regarding the kind of devices, our proposal relies on an element called *Mobility Framework Resolver* (MFR) for storing a matching table between the *(user identifier, device identifier)* and the *(IP address, port)* in case of regular devices or the *(platform specific identifier)* in case of mobile devices.

As shown in Figure 4, the interaction *update* allows any node to update its connectivity information to the MFR. The interaction *resolve* queries the MFR with a user and device identifier in order to obtain the actual IP address and port or the platform specific identifier. Due to restrictions in the *Messaging Service*, regular devices don't directly access that service. So, in case a regular device wants to communicate with a mobile device, the interaction *request* asks the MFR to send a message *forward* to the mobile device containing regular device's IP address and port. Therefore, the mobile device can communicate back with the regular device based on the message *report* from the MFR. Furthermore, the *MFR* acts as relay between devices in case of connectivity limitations.

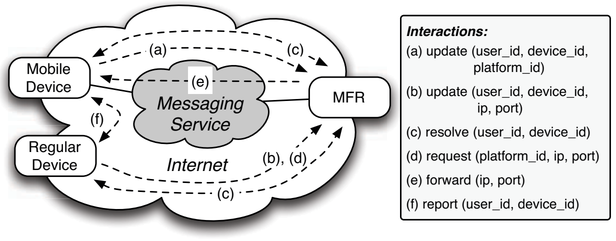

Mobile Device
(a)
(c)
(e)
Messaging Service
MFR
(f)
Regular Device
Internet
(b), (d)
(c)
Interactions:
(a) update (user_id, device_id, platform_id)
(b) update (user_id, device_id, ip, port)
(c) resolve (user_id, device_id)
(d) request (platform_id, ip, port)
(e) forward (ip, port)
(f) report (user_id, device_id)

Fig. 4. User Bindings defines the Mobility Framework Resolver (MFR) joint with a set of interactions for supporting mobility between networks at middleware level.

C. Mobility across devices

Sending a message to a user means sending a message to the device the user is currently using at that moment (i.e. where the User Component is active). Based on the schema shown in Figure 3, the middleware thus identifies that device with a reserved device identifier (*human*) in addition to the identifier assigned by the user. The procedure for supporting mobility between devices at middleware level is composed by the following steps: the destination device automatically notifies the origin one, adds the identifier *human* to its denomination and activates the User Component if available.

Nevertheless, mobility between devices is not fully accomplished at this moment as the user is supposed to move between devices without regarding the connectivity limitations or the software installed. Based on previous work [20], [21], using Online Social Networks (OSNs) at the middleware level to reach a user reduces the impact of those constraints, and hence improves the availability of the user. Therefore, our approach relies on OSNs to communicate with the user (a) when the device used by the human is not reachable (e.g. due to connectivity restrictions), (b) when the required software is not installed in the device, or (c) when the user is not using any device.

Lastly, our middleware supports the required granularity for communicating selectively either with the user or with any device using multiple communication channels for supporting both mobility between networks and devices.

V. Building Participatory Sensing Applications with @migo

We implemented three representative @migo-based participatory sensing applications: thermal comfort, heater monitoring and window state reporting. We focus on the following aspects: (i.) how to provide a 2-way information flow between occupants and building manager, (ii.) ensuring that sensors deliver information directly to the occupants and (iii.) dealing with context awareness for improving occupant experience. Additionally, the system also combines users' contributions with fixed infrastructure measurements (temperature, humidity and windows state sensor), as shown in Figure 5(b).

Thermal comfort application: The goal of this application is to obtain real time feedback from users as defined in [23]. An additional contribution on the overall feeling is also requested from the participants each day. Participants' contributions combined with the values from fixed sensors allow the system to prompt users to perform certain actions (e.g. open or close a window) in order to improve the experienced comfort level. In this application, humans are used as both opportunistic sensors and flexible actuators.

Heater monitoring application: The goal of the heater monitoring app is to measure the state of heaters in each room, so that the system can recommend appropriate actions to the participants in the thermal comfort app. As heaters are regulated by a manual dial, humans are used as opportunistic sensors and actuators at lower cost than automated solutions.

Window monitoring application: The purpose of the windows state reporting is to notify the user when the window is open outside of business hours. If the window is detected to be open, the system checks if any user is still in the office by introspecting the User Component or relevant participants. Then, the system prompts the participants to close the window. Hence, humans are used as flexible actuators in this application that increase security and environmental control.

As shown in Figure 5(a), each application defines its own set of roles (based on geographical user's position) and interaction allowed in each one. Our implementation relies on Google Services and WiFi as outdoor and indoor positioning mechanisms. For this purpose, each room is provided with a router beaconing a well-known value, so each participant could configure the system to automatic recognise its own room. Additionally, the system automatically disconnects those mechanisms that are not used in that moment (e.g. GPS is not accessed while the WiFi beacon is detected) to reduce the consumption of energy.

A. Stakeholders

We identify three stakeholders in our architecture: the *organisers* create the applications (i.e. role definition, users' interactions allowed in each role and list authorised participants) and schedule queries, the *participants* contribute with values (opportunistic sensors) and actions (flexible actuators), and the *framework manager* maintains the system (i.e. database server, web front end and participatory sensing server) and processes the results. Participants are requested to install the @migo middleware. Specific software for each application may be installed manually or automatically by the @migo middleware. Organisers and the Framework Manager are also required to run the @migo middleware.

B. System Architecture

The User Component together with the User Bindings allows @migo to support participatory sensing applications. Figure 5(a) shows the component diagram supporting these three applications. The User Component represents uniquely the user into all the participatory sensing application. Although it runs in all @migo-enabled devices, it is only active in the device the user is handling at each moment. The *Reconfiguration Entity* component performs all the reconfiguration actions according to the subset of allowed interaction for each role. The *Role Detection* component may determine the user's role based on values from sensors (e.g. GPS) or processed data (e.g. activity recognition algorithms). It establishes the user's role by modifying the property Application List in the User Component. Other application specific components might also run on the user's device (e.g. log 3G signal strength).

Each participatory sensing application defines its own set of graphical interfaces that communicates the User Component with the user. These interactions are platform dependant. In case of Android devices, the User Component starts an activity according with the type of interaction (request or notification). Reply from the user is sent back in a message through the broadcast channel containing extra information (e.g. timestamp of the request) to decouple the graphical interfaces from the User Component.

C. Interfaces

Once the application is deployed and running, each stakeholder interacts with the participatory sensing system (and vice-versa) via dedicated user interfaces. The user obtains a request or a notification from the system through the relevant app or through Twitter. Contributions are reported through the app or a web browser. Organisers requires a web browser for querying the participants or for checking the results. The framework manager controls the database and accesses the management console of each participatory sensing application.

VI. Evaluation

The @migo prototype extends the LooCI Middleware [19] to include the User Component and the User Bindings. It uses Twitter as the namespace for user identifiers and as communication channel to the users. Although other OSNs are also an option, Twitter provides the best support for automated and semi-automated applications.

Table II shows the footprint of the User Component, the User Bindings, the Mobility Framework Resolver together with the size of the @migo middleware. Low footprint ensures good performance even in low-end mobile devices. Note the JAR files contain all required libraries for its execution.

TABLE II. Footprint of @migo and its elements supporting participatory sensing applications

	User Component	**User Bindings**	**@migo**	**MFR**
JAR file	21185 bytes	549589 bytes	725482 bytes	57010 bytes
JAR file for Android	28014 bytes	738478 bytes	987870 bytes	-

Table III shows the line of codes (LoC), cyclomatic complexity (CC) [24] and footprint of the implemented components for each participatory sensing application. Note that 'App Logic' only refers to specific Java code for the Android platform without including XML files or pictures included for the GUI. It should be also noted that the heating and window applications reuse the role detection component of the comfort application and thus their footprint has reduced 39% and 43% respectively. Furthermore, as these components are reused, significant development effort is saved (32% and 37% of LoC respectively).

TABLE III. Reusability of components between applications reduces the footprint of them

		LoC	**CC (avg)**	**Jar File for Android**	**Footprint/ LoC Red.**
Thermal	Reconfig. Entity	654	8,57	18519 bytes	0% / 0%
Comfort	Role Detector	743	3,36	43107 bytes	
App	App Logic	1487	1,90	84611 bytes	
Heaters	Reconfig. Entity	467	6,29	17507 bytes	39% / 32%
App	App Logic	1065	1,81	50041 bytes	
Windows	Reconfig. Entity	439	5,70	17328 bytes	43% / 37%
App	App Logic	795	2,09	39724 bytes	

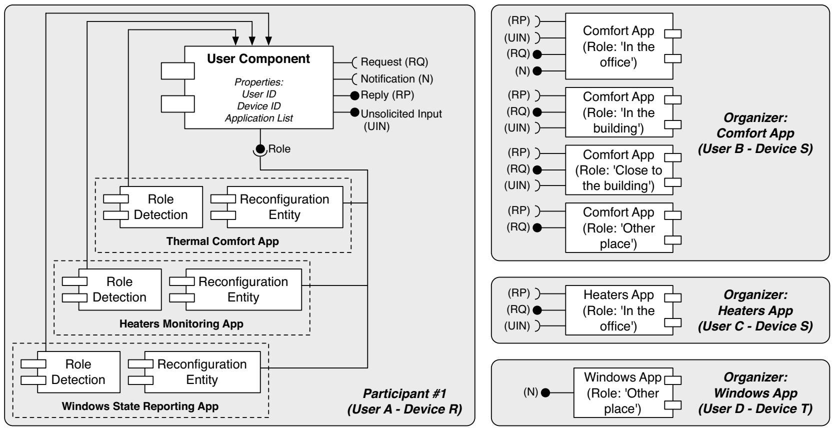
Table IV shows the size of the APK files that any participant downloads and installs in his Android mobile devices. The file also contains pictures and GUI specific code. Reduced size guarantees reduced downloading times even for low speed wireless connections (e.g. GPRS connections).

TABLE IV. Reduced size of the APK file of each participatory sensing application

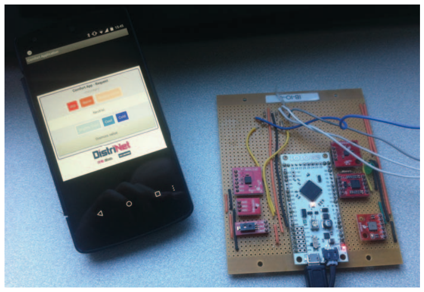
@migo	**Comfort App**	**Heaters App**	**Windows App**
3062 kB	527 kB	476kB	473kB

A. Avoiding Useless Messages

We evaluate our proposal by running the three applications (thermal comfort, heaters monitoring and windows state reporting) for 3 participants over two weeks. The system randomly generates 8 requests to participants with the roles *In the office*, *In the building* and *Close to the building*. Additionally, once per day, a request is sent to all participants asking for a general overview of the thermal comfort experience during that day. Other requests are sent on-demand to participants as required by the experimental organizer.


User Component
Properties:
User ID
Device ID
Application List
Request (RQ)
Notification (N)
Reply (RP)
Unsolicited Input (UIN)
Role
Role Detection
Reconfiguration Entity
Thermal Comfort App
Heaters Monitoring App
Windows State Reporting App
Participant #1 (User A - Device R)
Comfort App (Role: 'In the office')
Comfort App (Role: 'In the building')
Comfort App (Role: 'Close to the building')
Comfort App (Role: 'Other place')
Organizer: Comfort App (User B - Device S)
Heaters App (Role: 'In the office')
Organizer: Heaters App (User C - Device S)
Windows App (Role: 'Other place')
Organizer: Windows App (User D - Device T)

(a)



(b)

Fig. 5. (a) Component diagram supported by @migo. Each participant runs the *User Component* in addition to the application specific components *Reconfiguration Entity* and *Role Detection* (it modifies the property *Application List* in the User Component) and other application components. Organisers runs a component per context supporting with the allowed interactions. (b) Smartphone showing a request from the thermal comfort application close to fixed infrastructure sensors (temperature, humidity, windows state).

Supporting mobility between contexts through reconfiguration of the User Component avoid useless messages from being sent to the users. During the experiment, 446 unique requests were generated by the organisers. Only 219 of these were disseminated to the participants, while the rest were out-of-context and therefore discarded. This means that context-awareness saved participants from receiving useless requests in roughly half of the cases (50,90%).

B. Availability of users

Online social networks are used as a backup communication channel when standard application communication is unavailable. 76% (166 out of 219 requests) were delivered to the user's device through @migo. The remaining 24% (53 of 219) were delivered directly by Twitter. The use of OSNs therefore increased participant availability by 24%. Although using OSNs significantly increases the delivery time as shown in Table V, this remains well within the constraints of many participatory sensing applications.

TABLE V. DELIVERY TIME FOR EACH COMMUNICATION CHANNEL

	@migo	Twitter
avg	983,185 ms	5022,925 ms
stddev	612,293 ms	23,269 ms

C. Resolution of tasks

Considering only the replies that were received one hour after the request, 119 requests out of 219 requests (54%) were replied to during the experiment. This finding is in line with prior studies of user responsiveness by Crowley et al. [25]. Specifically, 109 (92%) of 119 replies were obtained through the native app for @migo and 10 replies (8%) through the web platform respectively. Therefore, offering multiple interfaces to the users increases participation of the users.

Figure 6 shows users response times for each communication channel. It reveals that Twitter also fits the requirements for participatory sensing applications.

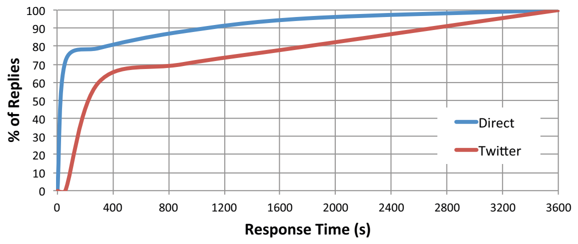


Fig. 6. Distribution of response time for each communication channel

D. Energy Consumption

We zoom into the energy consumption characteristics of the proposed solution using the Power Tutor software [26] for three Motorola Moto G smartphones running the comfort application.

As @migo and other applications in the smartphone rely on the positioning services offered by Google, it is not possible to determine which application is causing the energy consumption of the GPS. However, without regarding to GPS consumption, Table VI shows that @migo and the Thermal Comfort App has only reduced less than 1% and roughly 3% of battery lifetime respectively. Considering the worst case where all GPS energy consumption (16,033%) was caused by @migo, the experiment would have reduced battery lifetime by 19,241%.

TABLE VI. ENERGY CONSUMPTION (COMFORT APP)

	@migo	Comfort App	PowerTutor
CPU	0,030	0,011	3,989
LCD	0,022	3,119	0
3G	0,007	0	0
WiFi	0,015	0,004	0,067
Total (%)	0,074	3,134	4,056

The battery life improvements that arise from the reuse of context components will be a subject of a future experiment.

VII. Conclusion

Participatory sensing applications rely on humans for acting as opportunistic sensors and flexible actuators. This requires support for user dynamism including: mobility across networks, mobility between devices and context-awareness.

We presented the User Component and the User Bindings as essential elements of a comprehensive middleware for supporting participatory sensing applications. The former models the user as a first-class software component allowing developers to reconfigure and introspect using Component-Based Software Engineering tools, and hence, dealing with mobility between contexts. The latter allows the middleware to support multiple communication channels including OSNs to connect users with sensing applications. This approach also allows the application to selective communicate with the user or with a device by offering an identification scheme with multiple levels of granularity, and hence, dealing with mobility between networks and devices. In the context of smart buildings we implemented three participatory sensing applications for improving adaptive thermal comfort issues.

We have evaluated our approach by running a two week experiment involving the three applications and three users. The obtained results confirm that our approach increases user participation, reduce out-of-context interactions and decrease energy consumption by sharing context information between applications.

Our future work will include a more detailed assessment of energy. In the short term, we will identify the specific savings that arise from the reuse of context components. In the long term, we will also explore how cloud balancing techniques for saving energy can maximise battery lifetime. We also intend to undertake a deep security analysis in order to develop a security model for @migo. Finally, we will investigate the scalability of our approach through long lived experiments with more participants.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven and the iMinds IoT research program.

References

[1] N. Brouwers and K. Langendoen, "Pogo, a Middleware for Mobile Phone Sensing," in *Proceedings of the 13th International Middleware Conference (Middleware'12)*. Springer-Verlag, 2012.

[2] Y. Chon, N. D. Lane, F. Li, H. Cha, and F. Zhao, "Automatically Characterizing Places with Opportunistic Crowdsensing Using Smartphones," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp'12)*. ACM, 2012.

[3] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow, "SociableSense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing," in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking (MobiCom'11)*. ACM, 2011.

[4] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, and R. A. Peterson, "People-centric Urban Sensing," in *Proceedings of the 2nd Annual International Workshop on Wireless Internet (WICON'06)*. ACM, 2006.

[5] M. Demirbas, M. Bayir, C. Akcora, Y. Yilmaz, and H. Ferhatosmanoglu, "Crowd-sourced sensing and collaboration using twitter," in *IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM'10)*. IEEE, 2010.

[6] M. Paxton and S. Benford, "Experiences of Participatory Sensing in the Wild," in *Proceedings of the 11th International Conference on Ubiquitous Computing (UbiComp'09)*. ACM, 2009.

[7] K. Roberts, M. A. Roach, J. Johnson, J. Guthrie, and S. M. Harabagiu, "EmpaTweet: Annotating and Detecting Emotions on Twitter," in *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*. ELRA, 2012.

[8] C. Szyperski, *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press, 2002.

[9] "Android OS," http://www.android.com/.

[10] "Twitter," http://www.twitter.com/.

[11] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory Sensing," in *Workshop on World-Sensor-Web: Mobile Device Centric Sensor Networks and Applications (WSW'06)*. ACM, 2006.

[12] J. Payton and C. Julien, "Integrating Participatory Sensing in Application Development Practices," in *Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER'10)*. ACM, 2010.

[13] H. Tangmunarunkit, C. K. Hsieh, B. Longstaff, S. Nolen, J. Jenkins, C. Ketcham, J. Selsky, F. Alquaddoomi, D. George, J. Kang, Z. Khalapyan, J. Ooms, N. Ramanathan, and D. Estrin, "Ohmage: A General and Extensible End-to-End Participatory Sensing Platform," *ACM Trans. Intell. Syst. Technol.*, May 2015.

[14] G. Cardone, A. Cirri, A. Corradi, and L. Foschini, "The ParticipAct Mobile Crowd Sensing Living Lab: The Testbed for Smart Cities," *IEEE Communications Magazine*, Oct 2014.

[15] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "PRISM: Platform for Remote Sensing using Smartphones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*. ACM, 2010.

[16] I. Podnar Zarko, A. Antonic, and K. Pripužic, "Publish/Subscribe Middleware for Energy-efficient Mobile Crowdsensing," in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication (UbiComp '13 Adjunct)*. ACM, 2013.

[17] B. Balaji, H. Teraoka, R. Gupta, and Y. Agarwal, "ZonePAC: Zonal Power Estimation and Control via HVAC Metering and Occupant Feedback," in *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings (BuildSys'13)*. ACM, 2013.

[18] J. Hicks, N. Ramanathan, D. Kim, M. Monibi, J. Selsky, M. Hansen, and D. Estrin, "AndWellness: An Open Mobile System for Activity and Experience Sampling," in *Wireless Health 2010 (WH'10)*. ACM, 2010.

[19] D. Hughes, K. Thoelen, J. Maerien, N. Matthys, J. Del Cid, W. Horre, C. Huygens, S. Michiels, and W. Joosen, "LooCI: The Loosely-coupled Component Infrastructure," in *11th IEEE International Symposium on Network Computing and Applications (NCA'12)*. IEEE, 2012.

[20] R. Bachiller, D. Hughes, S. Michiels, and W. Joosen, "Users As Reconfigurable Elements in Distributed Sensing Applications," in *Proceedings of the 7th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks (MidSens'12)*. ACM, 2012.

[21] R. Bachiller, D. Hughes, S. Michiels, and W. Joosen, "Enabling Massive Scale Sensing with the @LooCI Mobile Sensing Framework," in *IEEE 15th International Conference on Computational Science and Engineering (CSE) (EUC'12)*. IEEE, 2012.

[22] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389, Oct. 2008.

[23] "ANSI/ASHRAE Standard 55-2010 - Thermal Environmental Conditions for Human Occupancy," ASHRAE, 2010.

[24] T. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, 1976.

[25] C. Crowley, W. Daniels, R. Bachiller Soler, W. Joosen, and D. Hughes, "Increasing user participation: An exploratory study of querying on the Facebook and Twitter platforms," *First Monday*, August 2014.

[26] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS'10)*. ACM, 2010.