Supporting the development of network-aware reactive applications on smartphones

Gloria Ciavarrini, Luciano Lenzini, Valerio Luconi, Alessio Vecchio
Dip. di Ingegneria dell'Informazione, University of Pisa, Pisa, Italy
e-mail: {gloria.ciavarrini, valerio.luconi}@for.unipi.it, {l.lenzini, a.vecchio}@iet.unipi.it

***Abstract*—In the last years, research about context-aware systems has been particularly intense. Nevertheless, most of the proposed approaches and systems failed to flow from research to industry. In this paper we propose ANARC, a library that eases the development of network-aware applications for smartphones. ANARC does not try to cope with all the possible meanings and variations of context, it instead focuses on a specific restriction: the network and associated properties. ANARC adopts a rule- and trigger-based approach: when the network context matches the one described in a rule, the corresponding notification is sent to the application level. Examples of use of the proposed library – for mapping network coverage, detecting roaming regions of mobile operators, and monitoring WiFi access points – are included to demonstrate the benefits of the proposed approach.**

I. Introduction

The personal nature of smartphones and the intimate relationship with their owners fueled the adoption of these nowadays ubiquitous devices as the de-facto platform for context-aware applications. In this area, research mostly concentrated on two directions: i) middleware systems to support the development of context-aware applications [1], [2] and ii) the definition of methods and techniques useful to represent and manage context-related information [3], [4], [5].

A popular definition of context is the one provided by Dey and Abowd [6]: any information that can be used to characterize the situation of an entity, where an entity can be a person, place or object that is relevant in the interaction between users and applications. Context-aware applications use such information to customize services and to provide a better user experience. In many cases, context-aware systems include reasoning functionalities, to deduce new and/or higher level information from raw data. The reader is forwarded to [2] for a survey about context-aware mobile networking and additional references on such topic, whereas a general background about context modeling and reasoning techniques can be found in [7].

Nevertheless, despite the large amount of research carried out in the last years and the ubiquitous diffusion of smartphones, widespread adoption of context-aware applications is still not a reality. One of the possible reasons for this slow transition from the research world to the industrial environment is, according to the authors, the lack of easy-to-use mechanisms for incorporating contextual information at the application level.

A. Background

Here we recall some of the most important systems that support programming and execution of context-aware applications on smartphones.

ContextPhone is a prototyping platform for context-aware mobile applications [8]. The design goal and philosophy of ContextPhone is to provide context as a resource and to enable rapid application development. To fill the gap between operating system functionalities and the needs of application developers, ContextPhone provides modules that abstract sensors, system services, and communication. ContextPhone was available for Symbian-based phones.

WhozThat is a system that ties together social networking and context-awareness with smartphones [9]. In this case, the interaction between the physical and the virtual world is bidirectional: when a user meets other people, the devices cooperate to discover their social network IDs, then information about users, downloaded from Facebook, LinkedIn, etc., is used to discover possible common interests. This information is also used to customize the physical context that surrounds them (e.g. to play music that is enjoyable for all of them).

Medusa is a programming framework for crowdsensing applications that provides mechanisms for acquiring sensor data from multiple smartphones [10]. In particular, with Medusa, recurrent crowdsensing operations can be compactly expressed using a high-level programming language. Further, a distributed runtime system for task execution provides coordination mechanisms between smartphones and cloud components. Support for concurrent and asynchronous execution enables high throughput in terms of tasks.

PRISM is a platform for remote sensing using smartphones [11]. The goal of PRISM is to provide a flexible substrate for participatory sensing applications on a smartphone-based platform. It allows developers to package their applications as executable binaries and automatically propagates the application code to an appropriate set of smartphones based on a set of predicates. Users who would like to participate have to install the PRISM runtime, a middleware that runs on top of an existing OS, on their smartphones and register their device with the PRISM infrastructure.

USense is a utility-based smartphone middleware for executing community-driven sensing tasks [12]. It provides client-side mechanisms, which enable a unified sensing architecture, and expressive constructs, which make possible

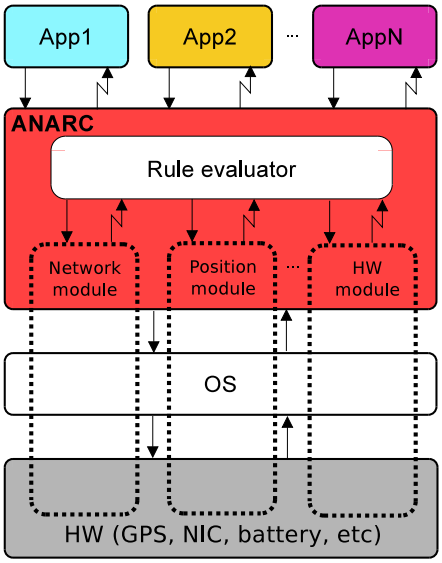


Fig. 1: ANARC architecture

efficient control and coordination of the sensor network. USense operates between applications and sensors and carries out crowd-sensing tasks by combining back-end application requirements, users' preferences, and available resources on smartphones.

B. Motivation

To fill the gap between network-aware applications and the functionalities provided at the OS level, we designed and implemented ANARC. *A*NARC is a *N*etwork *A*ware library for *R*eactive *C*omputing which, instead of facing the multiple meanings of context, operates in a single specific domain (network-awareness) and eases the development of network reactive applications. ANARC acts as a bridge between complex existing approaches and the real necessities of designers and programmers. In fact, a large fraction of existing systems are based on ontologies and provide sophisticate and expressive methods to describe context entities and infer knowledge. We believe that, in some cases, these mechanisms can be too far from the necessities of developers of smartphone apps, and that a library, which can be integrated with limited effort, can be useful to increase the context awareness of nowadays smartphone software.

Even though the primary goal of ANARC is to support the development of network reactive applications, it is important to notice that in several situations network-context can be used to gain more general information. For instance, if a smartphone is connected to a given access point during the night hours more or less regularly, this network information can be used to infer with reasonable confidence that such access point is located at user's home. Then, subsequently, visibility of the same access point can be used to infer that the user is at home and to trigger specific actions.

II. ANARC

ANARC is a runtime component that encapsulates all the intricacies related to the detection of network events and context changes, easing the production of network reactive applications. With ANARC, programmers specify the context properties of interest in a simple way, and receive notifications about relevant changes. Figure 1 describes the architecture of the system and its interactions with applications and lower layers.

Applications express their interest in receiving notifications about desired conditions and/or changes in network context by registering one or more *rules* to ANARC. Rules can be defined using a set of predefined properties. This set is mostly composed of network-related properties, since ANARC is devoted to support network-aware applications, but it also includes hardware-related and position-based properties since, in several cases, these appeared to be relevant in the considered reference scenarios. All these properties are exported and managed by specific subcomponents, which span the OS and hardware layers.

The *Rule evaluator* component parses the rules received from applications and then asks the appropriate domain-specific modules to start monitoring the involved properties and to notify about changes. When the conditions expressed by a rule are met, the evaluation component notifies the associated application, which in turn executes a registered handler.

A. Rules

Each rule is composed of the following parts:

- *Condition* It specifies the network context the application considers of interest. Context conditions are expressed as simple boolean expressions, where network properties can be combined through common boolean operators and some special functions.
- *Changing properties* Applications can receive notifications when a monitored property changes.
- *Action* This element specifies the type of notification the application wants to receive when the condition becomes true or when a monitored property changes its value. The possibility of associating a different action to every rule is particularly useful when registering more rules. In this way the application can react differently according to the triggering rule. This element can also include a list of properties whose value is transferred to the application when the action is triggered.

Several rules can be submitted at the same time to ANARC by enclosing them within a list. Table I shows a list that includes the prototype of three rules. Each rule is formed by a `when` element and by an `onChange` element. Both elements are optional: the first rule includes only the `when` element, whereas the second rule comprises only the `onChange` element.

`when` is in turn composed by three parts: i) `condition`, a boolean expression that specifies the interests of applications; ii) `doIfTrue`, that specifies the action that has to

TABLE I: A list of rules

```
<ruleList>
<rule>
 <when>
  <condition>
    boolean condition
  </condition>
  <doIfTrue> action </doIfTrue>
  <doIfFalse> action </doIfFalse>
 </when>
</rule>

<rule>
  <onChange>
   <property>
     property to be monitored
   </property>
   <do> action </do>
  </onChange>
</rule>

<rule>
 <when>
  <condition>
    boolean condition
  </condition>
  <doIfTrue> action </doIfTrue>
  <doIfFalse> action </doIfFalse>
  <onChange>
   <property>
     property to be monitored
   </property>
   <do> action </do>
  </onChange>
 </when>
 <propertyList>
  <property>
    property name
  </property>
  <property>
     ...
  </property>
 </propertyList>
</rule>
</ruleList>
```

be performed when the condition becomes true; and *iii*) `doIfFalse` that specifies the action to be performed when the condition becomes false. In particular, the specified actions are triggered each time the associated condition becomes true or false respectively.

`onChange` can be used to specify a property to be monitored (`property` element) and the action the application wants to receive when the property changes. The optional `threshold` element can be used for those properties that have continuous values. In this case the application is notified only if the observed variation is larger than the provided threshold.

If `onChange` is contained within a `when` element, as it happens in the third rule listed in Table I, then the property under observation is monitored by ANARC only when the associated condition becomes true. This allows programmers to combine the above mechanisms and limit the notifications an application receives to specific restrictions of the context.

The `propertyList` element is used to specify the set of properties whose values must be transferred to the application when the action is triggered. These values can be used by the application to customize its operations.

B. Properties

Table II lists some of the observable properties and context elements that can be combined to form a condition. Properties are classified according to their domain: network, hardware, and position (geography). The three domains are identified by the *net*, *hw*, and *geo* top-level prefixes. Other subdomains are used to group properties and provide a namespace-like mechanism. For instance, the *net.wifi.RSSI* property indicates the received signal strength indicator as registered by the WiFi interface, whereas the *net.cellular.RSSI* property concerns the received signal strength indicator as registered by the cellular interface.

ANARC provides a number of operators and pre-built functions that can be used to specify conditions. Operators are useful, for instance, to compare properties against thresholds or other values that developers consider of interest. Functions are used to perform some common operations that are difficult to express as combinations of logical and arithmetical operators (e.g., checking if an IP address is within a given range or verifying if current position belongs to a geographical area). Some of the available operators and functions are listed in Table III.

Table IV shows an example rule that tests if the device is using a data connection and if the signal strength is greater than a given threshold (6). When the boolean condition becomes false no action must be executed. On the contrary, when the boolean condition becomes true, the *it.unipi.iet.myaction* action is launched and monitoring of cell ID variations is started. The property list specifies the contextual values the application wants to receive whenever the action is triggered (in this case: latitude, longitude, type of connection, and operator name).

III. Implementation

As mentioned, the ANARC architecture comprises a *rule evaluator* and a set of *property monitoring* modules. At this time, three different property monitoring modules are available (one for each domain): the *network* module which monitors all the network properties, the *hardware* module which monitors the properties related to the smartphone's hardware, and the *position* module which monitors the properties related to the geographic position of the smartphone. When one of these components detects a change in one or more properties, it notifies the *rule evaluator* about the new value of the changed properties.

As the name suggests the *rule evaluator* component monitors state changes of entire rules, and fires the associated actions when needed. The core of the *rule evaluator* is a set of hash tables in which rules and properties are stored. Every rule submitted to ANARC is tagged with a unique Rule ID (RID). Each property is the key of a map which associates a property to all the RIDs of the rules containing that property. When a property monitoring module detects a change in the state of a property, it notifies such variation to the *rule evaluator*. For each rule associated to that property, the *rule evaluator* checks if the rule's condition becomes true/false or if an `onChange`

TABLE II: Some properties

Network domain	
net.cellular.gsm.cellID	Current GSM Cell ID (CID)
net.cellular.gsm.LAC	Current Location Area Code
net.cellular.networkOperatorName	Current network operator name
net.cellular.cdma.sysID	Current system ID for CDMA networks
net.cellular.cdma.netID	Current network ID for CDMA networks
net.cellular.cdma.bsID	Current base station ID for CDMA networks
net.ip.address	Current IP address assigned to the device
net.networkAccess.status	Status of network access (CONNECTED, CONNECTING, DISCONNECTED, etc)
net.networkAccess.connectivityType	Type of current network access (WIFI, MOBILE)
net.networkAccess.mobileConnectionType	Type of mobile connection (UMTS, HSUPA, EDGE, etc)
net.operator.roamingStatus	The device is in roaming (true, false)
net.cellular.RSSI	Current RSSI value
net.cellular.ASU	Current ASU value
net.wifi.RSSI	Current WiFi RSSI value
Hardware domain	
hw.wifi.state	Status of WiFi interface (DISABLED, DISABLING, ENABLED, etc)
hw.bluetooth.state	Status of Bluetooth interface (OFF, TURNING_OFF, ON, etc)
hw.data.state	Status of data connection (CONNECTED, CONNECTING, SUSPENDED, etc)
hw.battery.state	Status of battery (CHARGING, DISCHARGING, FULL, etc)
hw.battery.level	Current battery level value.
hw.battery.health	Health of battery (COLD, DEAD, GOOD, etc)
hw.battery.plugged	Charging type (AC, USB, etc)
hw.gps.status	Status of GPS receiver (FIRST_FIX, STARTED, STOPPED, etc)
Geographic domain	
geo.position.latitude	Current latitude value
geo.position.longitude	Current longitude value
geo.position.altitude	Current altitude value
geo.speed	Current speed value

TABLE III: Some functions and operators

Functions	
`ipRange(ip_to_check, ip_start, ip_end)`	Checks if *ip_to_check* belongs to a continuous IP address range and if it is between *ip_start* and *ip_end*.
`subnet(ip_to_check, subnet, mask)`	Checks if *ip_to_check* belongs to a given *subnet/mask*.
Operators	
`num_equals('numeric_value_one', 'numeric_value_two')`	Compares two numbers (equal).
`num_gt('numeric_value_one', 'numeric_value_two')`	Compares two numbers (greater than).
`num_lt('numeric_value_one', 'numeric_value_two')`	Compares two numbers (lower than).
`&&`	Logical AND
||	Logical OR

action must be launched. If so, it triggers the execution of the action associated to that rule.

The ANARC library pauses all unnecessary modules as soon as possible to reduce power consumption (a module can be paused when all the active rules do not involve any of the properties monitored by such component).

Currently, ANARC is available for the Android platform. Communication between applications and ANARC takes place using *Intents*, as it is usually done in the Android ecosystem. On submitting a rule to ANARC, the app receives an identifier. Such identifier can be used to deactivate and remove the rule if needed.

IV. Building network-aware applications with ANARC

In this section we illustrate some network-aware applications built using ANARC. The first two applications focus on cellular network coverage and detection of roaming regions. The third one is aimed at monitoring a set of WiFi access points.

A. Signal coverage maps

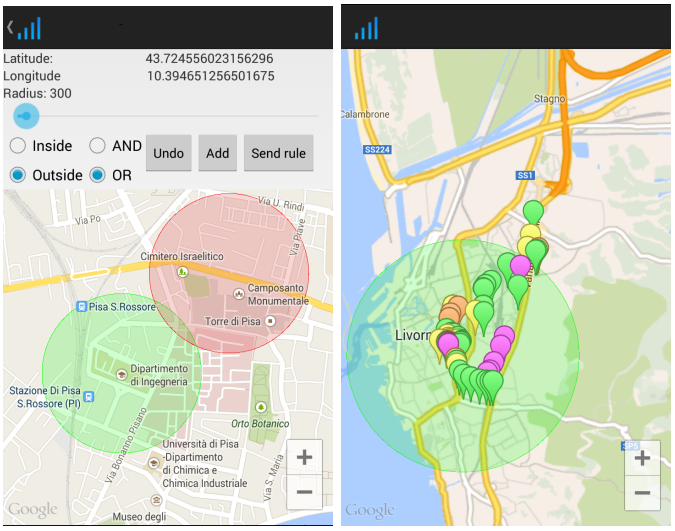
In the last years, several applications dedicated to mapping the coverage of cellular networks have been presented. Examples include Portolan Network Tools [13], [14], [15], RootMetrics CoverageMap [16] or NetRadar [17]. Typically these application rely on the contribution of users, which manually run measurements to collect georeferenced samples about signal quality. However, all these applications are affected by a common problem: there are areas that are oversampled (where the majority of users live) and areas that are not sampled at all (in general rural areas) [18]. Thus, in order to speed up the collection process and to make it more uniform, it would be desirable to automate the collection of samples in specific areas of interest (the undersampled ones) avoiding collection in already covered areas (the oversampled ones).

TABLE IV: An example: if the device is using a data connection and if the signal strength is greater than a given value, monitoring of cell ID is started and `myaction` is delivered to the application

```
<ruleList>
  <rule>
    <when>
      <condition>
        equals(net.networkAccess.ConnectivityType, 'MOBILE')
             && num_gt(net.cellular.ASU, 6)
      </condition>
      <doIfTrue>
        it.unipi.iet.myaction
      </doIfTrue>
      <onChange>
        <property>
          net.cellular.gsm.cellID
        </property>
        <do>
          it.unipi.iet.action_cell_id_has_changed
        </do>
      </onChange>
    </when>
    <propertyList>
      <property>
        geo.position.latitude
      </property>
      <property>
        geo.position.longitude
      </property>
      <property>
        net.networkAccess.mobileConnectionType
      </property>
      <property>
        net.cellular.networkOperatorName
      </property>
    </propertyList>
  </rule>
</ruleList>
```

With the help of ANARC we built an application that automatically collects ASU (Arbitrary Strength Unit - a value proportional to the received signal strength) samples when the smartphone is located within a specific area. Table V (in appendix) contains an example of the list of rules that the application submits to ANARC. The rules specify that the application has to be notified when the smartphone is in one of the three circular areas of interest. ANARC then returns the current position and ASU value.

In general, smartphone position can be obtained in two ways: either using the GPS or using a network-based location provider (which determines the position of the device using visible access points and/or cellular towers). In both cases, the OS returns the position expressed in terms of latitude, longitude, and position accuracy. These three values identify a circular area where the smartphone is supposed to be. As known, GPS is in general the best localization method, with an accuracy that is in the order of few meters. The network-based method instead can have an accuracy in the order of hundreds or thousands of meters. On the other hand, GPS is much more energy demanding with respect to the network-based provider. Thus, in order reduce the consumption of energy, when monitoring the smartphone's position, ANARC implements an optimization based on a hybrid approach. By default ANARC monitors the position using the network-based provider. When the smartphone gets close to the area of interest ANARC starts monitoring its location using the GPS. In particular, the GPS is started when the following condition is satisfied: $|D - R| \leq A$, where D is the distance between the center of the area of interest and the smartphone, R is the radius of the area, and A is the accuracy.



(a) Areas editor (b) Sample map

Fig. 2: An app for building signal coverage maps: selective collection in undersampled regions

Figure 2 shows some screenshots of the app dedicated to signal coverage mapping. In particular, Figure 2a shows the interface that allows the user to specify the areas of interest (as circular areas). It is also possible to specify areas that should be avoided during collection of samples. These user-generated specifications are translated in rules similar to the one in Table V and submitted to ANARC[1]. Figure 2b depicts a signal coverage map where colored markers express the quality of signal.

B. *Detection of roaming borders*

In cellular communications roaming refers to the possibility for a customer to use (almost) all the services provided by his/her mobile operator by using the network of another operator. This usually happens when a customer travels in a foreign country, but in some cases roaming can also take place within the customer's own country, e.g. when the operator he/she is subscribed to does not provide full coverage.

A mobile operator could be interested in knowing the actual borders of its network, i.e. the exact points where its customers switch to another operator. A possible solution is represented by an application that logs and georeferences all the transitions to another operator. This information could be then collected on a central server for further processing. The client-side of such system can be implemented using ANARC. The rules

[1]In a complete system these rules would be received from the central server where all user generated content is collected and aggregated. Having a global view, the server would able to detect oversampled and undersampled regions.

that implement the requested behavior are shown in table VI: whenever the smartphone enters in a roaming region, the app is notified with current geographic coordinates and the hosting operator.

The second rule returns the network operator name when the user is in his/her home network. This rule is needed to obtain the smartphone's default operator.

C. Automatic monitoring of urban WiFi networks

We used ANARC to implement an app for automatic monitoring of a urban WiFi network. The idea is to passively monitor a set of WiFi access points, all under the same administrative domain, as users wander within the urban area: every time an access point is found the app runs a set of network monitoring operations (e.g. measuring the delay or the bandwidth towards a given target). Collecting these statistics allows the network managers to detect overloaded access points or other problems.

The app has been customized to operate with *Pisa WiFi*, a municipality project that offers free Internet connection by means of a number of access points scattered all over the town. To use this service users need to log in. Account credentials are sent using an SMS after registration.

The app has been implemented using the rules shown in Table VII. The app is notified when i) a "Wi-Fi Pisa" access point becomes visible and ii) when the smartphone is connected/disconnected to/from such access point. In the first case, the app tries to connect to that Wi-Fi network searching the credentials among received SMSs or requiring user input. In the second case, the app starts an analysis task on connection, and stops running tasks on disconnection. One of the analysis tasks uses SmartProbe [19] for discovering the bottleneck capacity between two hosts.

V. Conclusion

As networked systems become more complex and ubiquitous, the adoption of programming paradigms based on network-awareness gets more important. ANARC allows programmers to design and implement network-aware applications without significant effort. Provided programming abstractions are deliberately simple and based on reactive approach. The presented applications show possible uses of ANARC and demonstrate that the system is sufficiently general and expressive to solve non trivial problems.

Acknowledgment

This work has been supported by the European Commission within the framework of the CONGAS project FP7-ICT-2011-8-317672.

References

[1] W. Xue and H. K. Pung, "Context-aware middleware for supporting mobile applications and services," in *Handbook of Mobile Systems Applications and Services*, A. Kumar and B. Xie, Eds. CRC Press, 2012.

[2] P. Makris, D. Skoutas, and C. Skianis, "A survey on context-aware mobile and wireless networking: On networking and computing environments' integration," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 1, pp. 362–386, 2013.

[3] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," *The Knowledge Engineering Review*, vol. 18, pp. 197–207, 9 2003.

[4] X. Wang, D. Q. Zhang, T. Gu, and H. Pung, "Ontology based context modeling and reasoning using OWL," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PerCom) Workshops*, March 2004, pp. 18–22.

[5] B. Y. Lim and A. K. Dey, "Design of an intelligible mobile context-aware application," in *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, ser. MobileHCI '11. New York, NY, USA: ACM, 2011, pp. 157–166.

[6] A. K. Dey and G. D. Abowd, "Towards a better understanding of context and context-awareness," in *In HUC 99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. Springer-Verlag, 1999, pp. 304–307.

[7] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive Mob. Comput.*, vol. 6, no. 2, pp. 161–180, Apr. 2010.

[8] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen, "ContextPhone: a prototyping platform for context-aware mobile applications," *Pervasive Computing, IEEE*, vol. 4, no. 2, pp. 51–59, 2005.

[9] A. Beach, M. Gartrell, S. Akkala, J. Elston, J. Kelley, K. Nishimoto, B. Ray, S. Razgulin, K. Sundaresan, B. Surendar, M. Terada, and R. Han, "Whozthat? evolving an ecosystem for context-aware mobile social networks," *Network, IEEE*, vol. 22, no. 4, pp. 50–55, 2008.

[10] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan, "Medusa: A programming framework for crowd-sensing applications," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 337–350.

[11] 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*. ACM, 2010, pp. 63–76.

[12] V. Agarwal, N. Banerjee, D. Chakraborty, and S. Mittal, "USense – A Smartphone Middleware for Community Sensing," in *Proceedings of the 14th IEEE International Conference on Mobile Data Management (MDM)*, vol. 1, June 2013, pp. 56–65.

[13] A. Faggiani, E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio, "Smartphone-based crowdsourcing for network monitoring: Opportunities, challenges, and a case study," *Communications Magazine, IEEE*, vol. 52, no. 1, pp. 106–113, January 2014.

[14] E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio, "Sensing the internet through crowdsourcing," in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, March 2013, pp. 248–254.

[15] A. Faggiani, E. Gregori, L. Lenzini, S. Mainardi, and A. Vecchio, "On the feasibility of measuring the internet through smartphone-based crowdsourcing," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2012 10th International Symposium on*, May 2012, pp. 318–323.

[16] "RootMetrics Coverage Map," http://www.rootmetrics.com/.

[17] "Netradar," http://www.netradar.org/.

[18] A. Faggiani, E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio, "Lessons learned from the design, implementation and management of a smartphone-based crowdsourcing system," in *Proceedings of the First ACM Workshop on Sensing and Big Data Mining*, ser. SenseMine '13. New York, NY, USA: ACM, 2013.

[19] F. Disperati, D. Grassini, E. Gregori, A. Improta, L. Lenzini, D. Pellegrino, and N. Redini, "Smartprobe: A bottleneck capacity estimation tool for smartphones," in *Proceedings of IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, Aug 2013, pp. 1980–1985.

APPENDIX

TABLE V: Rule for signal coverage mapping

```
<ruleList>
  <rule>
    <when>
      <condition>
        CircularArea(center_lat1, center_lon1, radius1) ||
        CircularArea(center_lat2, center_lon2, radius2) ||
        CircularArea(center_lat3, center_lon3, radius3)
      </condition>
      <doIfTrue>
        it.unipi.iet.action_entering_area
      </doIfTrue>
      <doIfFalse>
        it.unipi.iet.action_exiting_area
      </doIfFalse>
      <onChange>
        <property>
          net.cellular.gsm.cellID
        </property>
        <do>
          it.unipi.iet.action_cell_has_changed
        </do>
      </onChange>
    </when>
    <propertyList>
      <property> geo.position.latitude </property>
      <property> geo.position.longitude </property>
      <property> geo.position.accuracy </property>
      <property> net.cellular.ASU </property>
      <property> net.cellular.networkOperatorName </property>
    </propertyList>
  </rule>
</ruleList>
```

TABLE VI: Rules for detecting roaming regions

```
<ruleList>
  <rule>
    <when>
      <condition>
        !net.operator.roamingStatus
      </condition>
      <do>
        it.unipi.iet.action_my_operator
      </do>
    </when>
    <propertyList>
      <property>
        net.cellular.networkOperatorName
      </property>
    </propertyList>
  </rule>
  <rule>
    <when>
      <condition>
        net.operator.roamingStatus
      </condition>
      <onChange>
        <property>
          net.cellular.networkOperatorName
        </property>
        <do>
          it.unipi.iet.action_operator_has_changed
        </do>
      </onChange>
    </when>
    <propertyList>
      <property> geo.position.latitude </property>
      <property> geo.position.longitude </property>
      <property> geo.position.accuracy </property>
    </propertyList>
  </rule>
</ruleList>
```

TABLE VII: Rules for automatic monitoring of urban WiFi networks

```
<ruleList>
  <rule>
    <when>
      <condition>
        substring(net.wifi.listSSID,'wifi pisa')
      </condition>
      <doIfTrue>
        it.unipi.iet.action_wifi_pisa_available
      </doIfTrue>
      <doIfFalse>
        it.unipi.iet.action_wifi_pisa_unavailable
      </doIfFalse>
    </when>
  </rule>
  <rule>
    <when>
      <condition>
        equals(net.wifi.currentSSID, 'wifi pisa')
      </condition>
      <doIfTrue>
        it.unipi.iet.action_wifi_pisa_connected
      </doIfTrue>
      <doIfFalse>
        it.unipi.iet.action_wifi_pisa_disconnected
      </doIfFalse>
    </when>
    <propertyList>
      <property> net.ip.address </property>
    </propertyList>
  </rule>
</ruleList>
```