SWAN-Fly : A flexible cloud-enabled framework for context-aware applications in smartphones

Roshan Bharath Das, Aart van Halteren, Henri Bal
Dept. of Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
{r.bharathdas, a.t.van.halteren, h.e.bal}@vu.nl

***Abstract*— Smartphones are equipped with various hardware sensors to enrich the user experience. SWAN is a middleware framework that supports easy collection and processing of sensor data. However, the limited resources of the smartphones prevent the apps from supporting big data applications that need to store and operate on large historical datasets, leading to the need for implementing a cloud solution. To this end we present the SWAN-Fly framework, a generic and flexible mechanism to ease the application developers' task of sending sensor data from the device to their preferred cloud solution for additional storage and processing. We evaluate the ease of use of SWAN-Fly on a Crowd Monitoring application and the flexibility is tested on two different cloud solutions.**

I. INTRODUCTION

There has been a growth in the number of onboard sensors in smartphones, such as the newly added fingerprint sensor in Samsung phones, the step sensor in Nexus or the 3D touch sensor in the iPhone. Many context-based applications can be created using these sensors, in a variety of domains such as healthcare, social networks, safety, transportation and environmental monitoring. To build such sensor based applications, developers need to have thorough knowledge of various sensor interfaces and expertise in sensor data collection, processing and evaluation. To provide better abstraction and programming support, some middleware frameworks such as SWAN[12], [13], Caredroid[14], Mobicon[16] and Sense-OS[6] have been proposed. Some systems send all sensor data to a remote cloud, which is undesirable for real-time interactive applications. Most other frameworks store all data on the phone, but this approach cannot deal with large sensor data sets. An ideal solution should support both flexible processing on the phone and efficient data transfer to the cloud. We describe below two application scenarios that would require such a solution:

- Computation on large historical data sets - Since smartphones have limited resources, it becomes hard to do processing over a longitudinal data set. Such data sets can be used in data analytics to understand usage statistics and in data mining to build prediction models.
- Distributed measuring - Many contextual applications in the area of distributed measuring require sensor data to be stored in a remote server. A good example is a Crowd Monitoring app that determines the live crowd situation in an area based on the GPS sensor data available from users in that area. Similarly, Google shows Popular times[8] based on historical visits to a location.

In addition, such a solution should give the application developers detailed control over what data goes to which cloud. As various clouds provide a multitude of services, it would make sense to connect to a cloud that is relevant for an app's need. For example, some apps require a specific cloud database (e.g., Shazam[1] exploits the audio sensor to identify the media being played, using a music database). Other apps require a cloud with much compute capacity, for example to run a large-scale simulation. Also, some apps want to have control over the privacy and security of the sensitive sensor data sent from the phone. Since apps get sensor data directly from a framework such as SWAN (being installed in the same device), it is desirable to let the application developers configure their endpoint parameters in SWAN so that the relevant sensor data can be sent to the corresponding cloud providers.

In this paper, we will therefore describe a flexible integrated solution that allows SWAN developers to easily store data from sensors in SWAN to their preferred cloud solution.

The main contributions of this paper are:

- A Generic Server Connection module - We created a generic server connection module to connect to any REST-based cloud solution of the application developers' choice.
- Extended SWAN-Song language - We extended the domain specific language for the application developers to easily add the endpoint parameters to an expression. We also built a preference screen to allow developers to easily tune the endpoint parameters and build the SWAN-Song expression.

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the SWAN framework and Section 4 presents the design and implementation of SWAN-Fly framework. In Section 5 we evaluate SWAN-Fly based on ease of use and flexibility. We conclude and outline our future work in Section 6.

II. Related Work

This section describes several frameworks which address the issue of storing and processing information from sensors on the device versus in the cloud. While most of these systems have chosen a solution where all the data is kept locally in the smartphone, others adopted a cloud based model where all the data is subsequently sent to a server for further processing.

Seemon[15] proposes a context monitoring framework for sensor-rich and resource-limited mobile environments where multiple applications can use a context monitoring query to understand the users context and react appropriately. The Mobicon[16] framework provides an application programming interface (API) and a runtime environment for applications. It functions as a shell above personal-area networks, providing an interface for receiving and processing sensor data and for controlling sensors. Neither of these solutions discuss the requirement for a cloud provider in the framework.

Fakoor, et al. [18] propose an integrated framework for storing, processing and delivering sensor data for people-centric applications deployed in the cloud. The smartphone will act as a thin client and is connected to a user adaptation module in the cloud that will deliver the sensor data to the relevant cloud engine in the cloud. Our work aims at covering all types of context-aware applications that do processing and storage of the sensor data in both the device and the cloud.

Rowlands, et al. [17] describe the need for a system to collect, store and analyse long-term sensor data in a multiple user context and discuss the design concepts to link sensors with cloud-based storage. MobiSens[19] is a client/server system with Android App as client and two-tier server systems. The mobile client collects various sensor data from phones, applies activity segmentation and lightweight adaptive activity recognition, and directly interacts with the user. Sensor data is uploaded to the MobiSens server to index and process the sensor data using heavy-weight algorithms. Second-tier (application and service) servers use information processed by the MobiSens server to provide application specific services such as lifelogging, senior care and ground reporting, etc. Sensingkit[20] and Sense-OS[6] provide a multi-platform library to collect sensor data from the phone and send it to the server. Such solutions bring tight coupling between the device and the server. Our approach focuses on helping the SWAN application developer to easily choose her preferred cloud provider to send the sensor data from the smartphone.

III. Background

In this section we describe the architecture of SWAN. More detail is provided in [12]. SWAN is a framework for building context-aware applications for the Android platform. It is designed to run as a background service that can be accessed by other applications through an API. SWAN offers powerful context abstractions to developers who would otherwise use the Android API for managing every sensor. It already contains 20+ sensors and supports plug-ins from third-party sensors. SWAN eliminates processing and storage redundancy by providing a centralized solution for collecting and storing the data from sensors on the phone, external sensors connected to the phone (e.g., by Bluetooth) and software sensors (e.g., open data on the Internet about the weather or traffic).

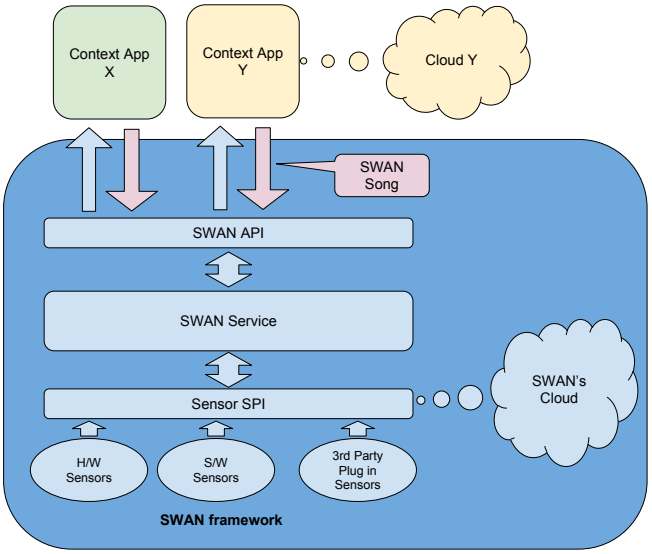


Fig. 1: Architecture of cloud solutions for SWAN

Fig. 1 shows the architecture of the SWAN framework. We will discuss the main components below:

A. SWAN- Song and SWAN API

Context aware applications register and unregister sensor expressions using the SWAN API. These sensor expressions are constructed using SWAN-Song, a domain specific language that allows application developers to easily deal with sensors on a high level.

There are two types of expressions:

1) Value Expressions: They return a list of values (data from sensors). For example, to get light sensor data we register the expression:

$$self@light : lux\{MAX, 1000ms\}$$

where $self$ is the location of the device, $light$ is the sensor name, lux is the value path, MAX represents the history reduction mode and $1000ms$ represents the history window. This expression will obtain the maximum of the values generated by the light sensor in the period of the last 1000 milliseconds.

2) Tri State Expressions: The app users can register a combination of multiple expressions to get a result in the form of TRUE, FALSE or UNDEFINED specifying if one or a combination of sensors meet the specified requirements. For example, the expression below checks whether the screen was off during the past hour and the battery percentage difference was more than 25%. This expression can be used to check if the battery consumption

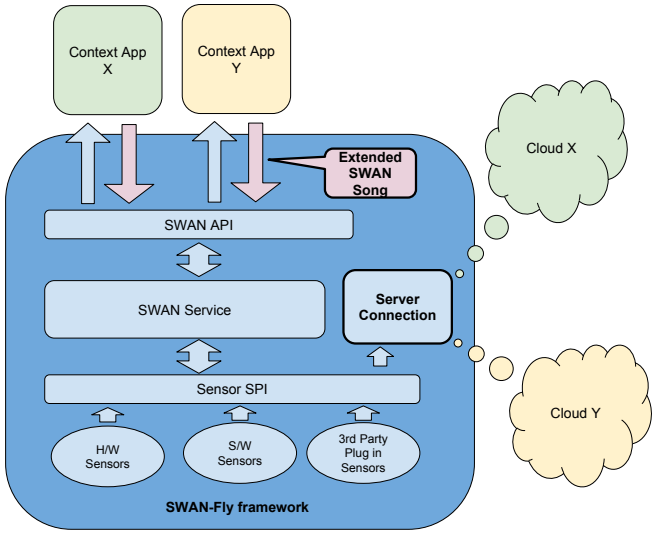


Fig. 2: Architecture of SWAN-Fly

is due to any process running in the background.

$$self@screen : on\{ALL, 1h\} == false \&\& \\ (self@battery : level\{MAX, 1h\} - \\ self@battery : level\{MIN, 1h\}) > 25$$

B. SWAN Service

The SWAN service bridges the gap between the sensors and the context expressions. It is a long-running background service that is responsible for starting the SWAN sensors and evaluating the registered expression on every new sensor event. After the evaluation, it will notify the changes in the result by sending broadcast messages (Android "Intents"[10]) to the app.

C. SWAN Sensors and Sensor SPI

SWAN sensors are background services that gather data from a particular resource, such as hardware sensors (on-device sensors like GPS, accelerometer, light, battery or sensors in external devices like heart rate monitor) or software sensors (weather data, train information data, etc). Application developers can easily add third-party sensors as a plug-in to SWAN using the Sensor Service Provider Interface (SPI). All sensor services run in separate processes, so if the SWAN service crashes for any reason, the sensor services will remain stable while the affected components recover. This prevents data from being lost. The communication between any sensor service and the other components is performed using inter-process communication.

IV. Design and Implementation

In this section we first describe how the SWAN framework was initially used to connect to a cloud provider. We then explain our implementation of a cloud solution integrated in the SWAN framework. We further discuss the disadvantages of the above solutions and we present the design and implementation of our framework, SWAN-Fly.

A. SWAN with external cloud solution

The context-aware apps can easily collect the sensor data from SWAN. In Fig. 1, the $Context\ App\ Y$ registers an expression in SWAN and receives the relevant sensor data. This data is then sent to the $Cloud\ Y$ for which the HTTP connection module is written by the developer. It implies that all the applications that need to send data to the cloud provider will have to build their own server connection module. SWAN's purpose to help developers to easily build context applications is not met in this case.

B. SWAN with integrated cloud solution

Our group built a cloud solution[23] for SWAN using a layered storage mechanism to store sensor data in the cloud. On receiving new sensor data, the sensor service will initially store the sensor data in the memory. It is then flushed to the local database from where it is further periodically transferred to SWAN's cloud. In Fig. 1, an expression from the $Context\ App\ X$ is registered in SWAN and the sensor data generated is periodically transferred to $SWAN's\ Cloud$.

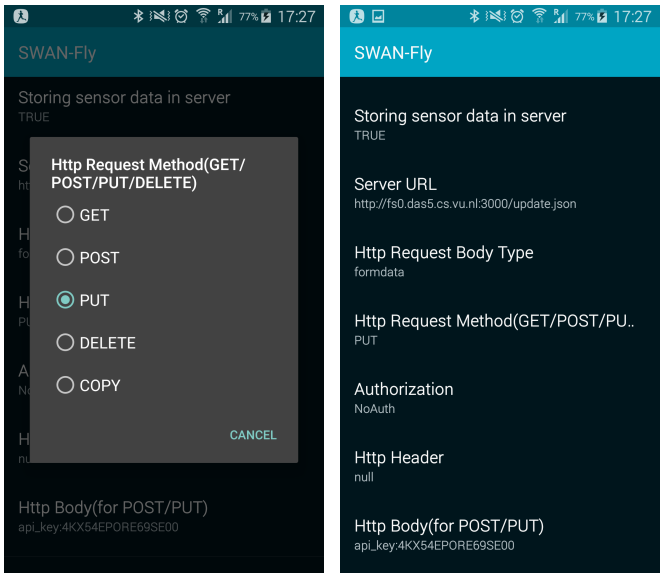
However, this approach gives tight coupling between the SWAN framework and the cloud provider. In some cases, the application developer would prefer to do additional processing which may not be supported by SWAN's cloud. In such cases, even though SWAN has an integrated cloud solution, the developer still needs to additionally build her own server connection module to send data from the SWAN's cloud to her own cloud solution.

C. SWAN-Fly framework

The SWAN-Fly framework decouples SWAN and the cloud solution so that developers can easily build context-aware applications that require sending data to their preferred cloud solution. To this extent, we introduce a generic REST based server connection module in SWAN that can be used to send sensor data to any server specified by the application developer. Moreover, we extend the SWAN-Song language by adding extra parameters for enabling data transfer to a remote server. The architecture is shown in Fig. 2.

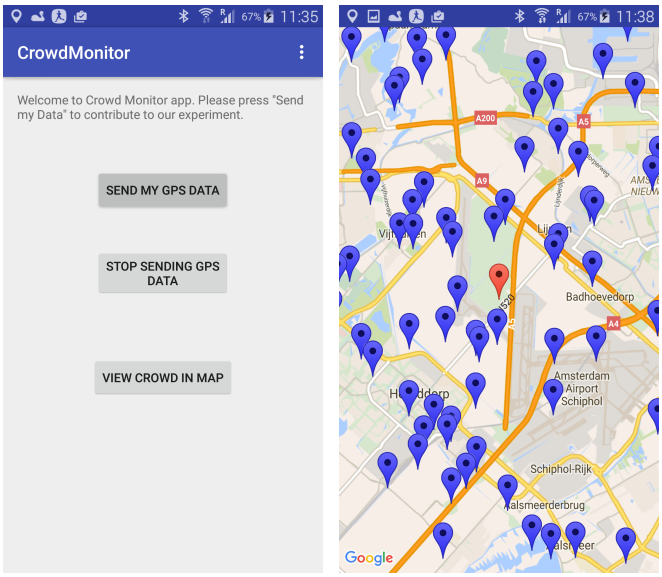
The following components are used for enabling a cloud solution in SWAN:

- Server URL to which the application developer wants to send the data.
- The request-body type for sending the data. For example, form-data, application/json, application/xml etc.
- The type of authorization (e.g. NoAuth, Basic Auth)
- The HTTP request method such as GET, POST or PUT. In case of GET method, the sensor data is encoded in the URL.
- A list of key-value pairs in the header.
- Apart from the sensor data generated, the developer can add extra fields in the HTTP body in the form of list of key-value pairs.
- Spatio-temporal information - In the Internet of Things context, since all data is represented in the form of



(a) Adding HTTP request method (b) Configuration parameters

Fig. 3: Server Connection Configuration



(a) Main Activity Screen (b) Crowd Map Screen

Fig. 4: Crowd Monitoring Application

space and time, the application developer can optionally enable the location and time fields for all sensors.

The sensor data will automatically be added to the HTTP body (in case of POST or PUT) in the form of key-value pairs where the developer can suggest the name of the key and the value is generated by the SWAN system.

Multiple context applications register expressions using the extended SWAN-Song domain specific language. To connect to a cloud, every expression should contain the endpoint (server) parameters. An example of registering an expression that uses light sensor data in the phone which is sent to the ThingSpeak[7] server is shown below:

$$self@light : lux\#endpoint_1\{MAX, 1000ms\}$$

where $self$ is the location of the sensor, $light$ is the entity name, lux is the valuepath, $endpoint_1$ is a set of server configuration parameters, MAX is the history reduction mode and $1000ms$ is the time period.

The endpoint $endpoint_1$ is configured as shown below:

$$\begin{aligned} &endpoint_1 = [\\ &url = https://api.thingspeak.com/update.json,\\ &http\_auth = NoAuth,\\ &http\_header = null,\\ &http\_body = api\_key : xx,\\ &http\_body\_type = formdata,\\ &http\_method = PUT] \end{aligned}$$

where url is the URL of the server which the developer wants to connect to, $http\_auth$ is the authorization which is no authorization in this case, $http\_header$ is the HTTP Header with $null$ value, $http\_body$ is the additional HTTP Body in which we set an API key for this sensor value, $http\_method$ is the HTTP Method which is PUT in this case.

To ease the creation of an expression, we have implemented a preference screen in SWAN as shown in Fig. 3 where the developer can tune the connection parameters.

In the above case, the light sensor service will start generating data. The Sensor SPI will check if the server configuration parameters are set. If so, the data is sent to the server connection module. A new connection is established for every registered expression and the sensor data is sent to the relevant endpoint. In Fig. 2 we can see that the data generated by $Context\ App\ X$ is sent to the $Cloud\ X$ and the data generated by $Context\ App\ Y$ is sent to the $Cloud\ Y$. Note that an app can also send sensor data to multiple cloud providers.

V. Evaluation

We evaluate SWAN-Fly using two criteria, namely the ease of use and the flexibility. The ease of use is essentially measured based on the number of lines of code the developer needs to write. We built a crowd monitoring application to compare the ease of use of the SWAN and SWAN-Fly frameworks. The flexibility aspect is evaluated based on how capable the SWAN-Fly framework is to connect to two different servers with different input data. We use ThingSpeak[7] and Django[4] servers to evaluate it. We will describe both cases below.

A. Ease of use

The crowd monitoring app is a prototype app built to detect the crowd density in a particular area of the user's interest in real time. Fig. 4 shows a screenshot of the app. The user can choose to send her GPS data to the cloud provider and view the live crowd density in the area of interest near her location. We built two versions of the crowd monitoring

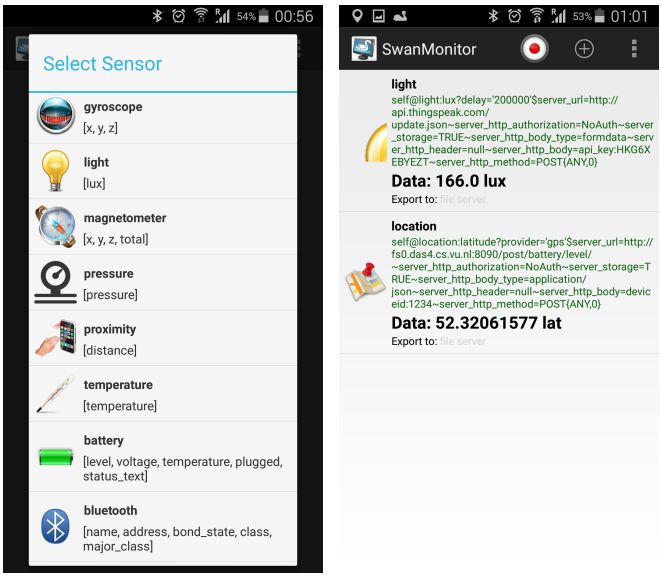
app that use SWAN and SWAN-Fly framework and we tested them using a Samsung Galaxy S5 running Android version 5.0.

The app using the original SWAN framework registers the GPS expression when the user presses the ”SEND MY GPS DATA” button. After evaluation, the SWAN framework sends GPS data to the app. The app uses the Android API to build the server connection module. The Android components used are the HttpURLConnection[9] client for sending and receiving data using HTTP and AsyncTask[11] to perform network operations on a separate thread from the UI thread. In the case of the app using the SWAN-Fly framework, the GPS expression along with the endpoint is registered and the live GPS data is sent to the endpoint from the SWAN-Fly framework.

We compare the number of lines of code for both versions as shown in Table I. It does not include white spaces, comments or logs and the maximum number of characters per line is 100. It is evident that the Crowd monitoring app using SWAN-Fly saves the application developer from writing extra lines of code. The SWAN-Song expression with SWAN-Fly uses extra lines of code to add information about the endpoint. With the use of the preference screen previously described, we note that it is easy to build SWAN-Song expressions.

TABLE I: Lines of code that the application developers need to write

Functionality	Crowd Monitoring app using SWAN	Crowd Monitoring app using SWAN-Fly
SWAN-Song expression	2	10
SWAN API	28	28
HttpUrlConnection	44	0
AsyncTask	15	0
Other (Non-SWAN)	132	132
Total	221	170



(a) Sensors in SWAN (b) Registered Expressions

Fig. 5: SwanMonitor Application

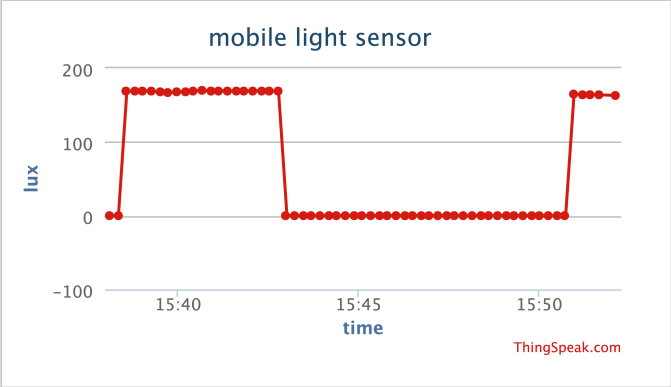


Fig. 6: Real time analytics in ThingSpeak

B. *Flexibility*

We test the flexibility of the SWAN-Fly framework by using the SwanMonitor[1] tool and two different cloud providers, namely ThingSpeak and Django server.

SwanMonitor is a utility app used to connect to SWAN using a graphical interface. It uses the SWAN API to get all sensors that SWAN provides, and the user can register multiple SWAN-Song expressions. The sensor data will be sent to the respective cloud provider mentioned in the expression. A screenshot of SwanMonitor is shown in Fig. 5.

ThingSpeak[7] is an open data platform for the Internet of Things. It supports real-time data collection, data analysis, data processing of the position information, data visualization, message transmission, etc. It provides an Open API to store and retrieve data from sensors using HTTP over the Internet or via a Local Area Network. Fig. 6 shows the real-time analytics of light sensor data collected from the SwanMonitor. The Y axis is the light intensity (lux) and the X axis is the $time$ at which the data was collected. It is interesting to note that we can infer various information with such analytics. In this case the 0 lux value implies the device was in a dark area and the 178 lux value implies that it was in a bright area. This could mean that in the time period from $15:43$ to $15:52$, the user had kept the device in her pocket. We can analyse the user behaviour in a work area with such information. The light sensor data is sent to the ThingSpeak server using the HTTP POST request with the request body type $form-data$ and api_key as additional HTTP body.

We deploy the Django[4] web framework on DAS4[2] and use the Apache Cassandra[5] database for storing the sensor data. DAS4 provides a common computational infrastructure for researchers who work on various aspects of parallel, distributed, grid and cloud computing, and large-scale multimedia content analysis. In this case, the DAS4[2] system can be used to do compute-intensive tasks such as simulations of a earthquake based on the users location data. With the use of the SwanMonitor tool, we send GPS sensor

[1]https://github.com/swandroid/SwanMonitor

data from the device to the server. The location data is sent to the Django server using HTTP POST request with the request body type as $application/json$ and the unique device id as additional HTTP body. In addition, we have deployed a private ThingSpeak server instance on DAS5[3], which allows us to keep our data on our own resource for additional cloud-based scenarios.

These experiments show that SWAN-Fly can be used for different REST based cloud providers by changing only the endpoint parameters in the extended SWAN-Song language.

VI. Conclusion and Future work

We designed and implemented a generic server connection module to send data to any cloud solution of the application developers' choice. We extended the SWAN-Song language to support easy configuration of expressions for enabling data transfer to any remote server. We evaluated the ease of use of SWAN-Fly with a Crowd Monitoring application. We further used the SwanMonitor tool to send various sensor data to multiple cloud providers and validated its flexibility. We note that SWAN-Fly, together with cloud-based processing frameworks (such as Cuckoo[22]), can enable a multitude of context-aware applications.

Our future work will focus on adding settings in the server connection module such as sync rate (send data every x seconds) and network connectivity (send only when connected to WiFi). We also plan to make the server connection module more generic so that all types of servers can be covered. To this extent we want to add lightweight protocols such as CoAP, MQTT etc. Further we will extend the SWAN-Song language to support easy fetching of data from the server. With this extension the developer will be able to provide the source of the data (in this case the endpoint URL) and the relevant HTTP connection parameters to get the data.

ACKNOWLEDGMENT

This work is funded by the municipality of Alkmaar, The Netherlands. It takes place in the context of Data Science Alkmaar and the Amsterdam Center for Business Analytics, led by Frans Feldberg. Kees Verstoep helped us with setting up the required components in the DAS5 system.

References

[1] Shazam, http://www.shazam.com/
[2] DAS-4 (The Distributed ASCI Supercomputer 4), http://www.cs.vu.nl/das4/
[3] DAS-5 (The Distributed ASCI Supercomputer 5), http://www.cs.vu.nl/das5/
[4] Django web framework, https://www.djangoproject.com/
[5] Apache Cassandra, http://cassandra.apache.org/
[6] Sense-OS, http://developer.sense-os.nl/
[7] ThingSpeak, https://thingspeak.com/
[8] Google Popular Times, https://support.google.com/business/answer/6263531?hl=en
[9] Android HttpURLConnection API, http://developer.android.com/reference/java/net/HttpURLConnection.html
[10] Android BroadcastReceiver API, http://developer.android.com/reference/android/content/BroadcastReceiver.html
[11] Android AsyncTask API, http://developer.android.com/reference/android/os/AsyncTask.html
[12] Kemp, R., 2014. Programming Frameworks for Distributed Smartphone Computing.
[13] Van Wissen, B., Palmer, N., Kemp, R., Kielmann, T. and Bal, H., 2010. ContextDroid: an expression-based context framework for Android. Proceedings of PhoneSense, 2010.
[14] Elmalaki, S., Wanner, L. and Srivastava, M., 2015, September. CAreDroid: Adaptation Framework for Android Context-Aware Applications. In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (pp. 386-399). ACM.
[15] Kang, S., Lee, J., Jang, H., Lee, H., Lee, Y., Park, S., Park, T. and Song, J., 2008, June. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In Proceedings of the 6th international conference on Mobile systems, applications, and services (pp. 267-280). ACM.
[16] Lee, Y., Ju, Y., Min, C., Yu, J. and Song, J., 2012, June. MobiCon: Mobile context monitoring platform: Incorporating context-awareness to smartphone-centric personal sensor networks. In Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on (pp. 109-111). IEEE.
[17] Rowlands, D., Ride, J., McCarthy, M., Laakso, L. and James, D., 2012. Linking sensor data to a cloud-based storage server using a smartphone bridge. In SENSORCOMM 2012, The Sixth International Conference on Sensor Technologies and Applications.
[18] Fakoor, R., Raj, M., Nazi, A., Di Francesco, M. and Das, S.K., 2012, August. An integrated cloud-based framework for mobile phone sensing. In Proceedings of the first edition of the MCC workshop on Mobile cloud computing (pp. 47-52). ACM.
[19] Wu, P., Zhu, J. and Zhang, J.Y., 2013. Mobisens: A versatile mobile sensing platform for real-world applications. Mobile Networks and Applications, 18(1), pp.60-80.
[20] Katevas, K., Haddadi, H. and Tokarchuk, L., 2014, September. Poster: Sensingkit: A multi-platform mobile sensing framework for large-scale experiments. In Proceedings of the 20th annual international conference on Mobile computing and networking (pp. 375-378). ACM.
[21] Palmer, N., Kemp, R., Kielmann, T. and Bal, H., 2012, November. SWAN-song: a flexible context expression language for smartphones. In Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones (p. 12). ACM.
[22] Kemp, R., Palmer, N., Kielmann, T. and Bal, H., 2010. Cuckoo: a computation offloading framework for smartphones. In Mobile Computing, Applications, and Services (pp. 59-79). Springer Berlin Heidelberg.
[23] Sovaiala, Nadina, et al. "A layered storage design for SWAN systems" Master thesis, Vrije Universiteit Amsterdam, 2014., https://www.academia.edu/s/865948e7e9