BeTelGeuse:
A Platform for Gathering and Processing Situational Data

BeTelGeuse is an extensible data collection platform for mobile devices that automatically infers higher-level context from sensor data. The authors introduce the BeTelGeuse architecture and evaluate its impact on mobile phone performance.

Joonas Kukkonen, Eemil Lagerspetz, Petteri Nurmi, and Mikael Andersson
Helsinki Institute for Information Technology HIIT

The widespread availability and portability of mobile phones has led them to become the de facto platform for ubiquitous computing. As mobile phones' battery life and capabilities continue to grow, they're supporting increasingly complex applications that leverage information about a user's situation—their location, activity, and so on. Modern smart phones are especially well-suited to this task because they're often integrated with sensing devices that facilitate obtaining detailed and meaningful descriptions of a user's situation. For example, smart phones can use accelerometers and microphones to accurately determine user activity[1] and can use Global System for Mobile Communications (GSM), WiFi, and GPS capabilities to determine users' locations and provide meaningful descriptions of their situations.[2]

To facilitate the gathering and processing of sensor data, we've developed BeTelGeuse, an open source platform that supports collecting data from phone sensors, Bluetooth-enabled sensors, and Internet data sources. BeTelGeuse also infers higher-level context from sensor data, for example, by inferring user activity from accelerometer measurements using the activity plug-in. Contrary to existing tools, BeTelGeuse isn't limited to a specific runtime environment or to a specific set of sensors. We designed BeTelGeuse to be extensible, as well as easy to use and configure. It's freely available under the GNU Lesser General Public License from our Web site (http://betelgeuse.hiit.fi). In this article, we present BeTelGeuse's design goals and architecture and evaluate its performance.

Design Goals

Our design goals for BeTelGeuse are:

- *Multiplatform support.* As we discuss in the "Related Work" sidebar, existing platforms are typically limited to a specific runtime environment. This limits the studies that researchers can carry out because the study's participants will depend on the number of available devices. To enable large-scale studies, the data collection platform should run on different devices and runtime environments.
- *Extensibility.* New kinds of sensing devices and data sources are continuously becoming available so researchers must be able to easily extend the platform to support them. Moreover, the platform's sensor interface shouldn't be limited to a specific type of sensor connectivity, such as Bluetooth, 802.11, or integrated sensors.
- *Data accessibility.* A platform that collects context data can provide applications with context information so it should be easy to integrate it with applications and services. Moreover, if researchers use the platform to run user studies, they should be able to access data remotely without requiring access to the physical device.

Published by the IEEE CS ■ 1536-1268/09/$25.00 © 2009 IEEE

Related Work in Data Collection Platforms

We categorize existing data collection platforms based on the nature of data that they collect. Platforms that collect objective data are nonintrusive as they gather sensor data about users' actions and situations without user involvement. The advantages of objective data are that users don't have to be interrupted and data collection doesn't suffer from subjective interpretations or from recall failures. On the other hand, sensor data is often noisy and erroneous, and unable to convey meaningful information about the users' situational, motivational, or informational needs. To this end, many platforms increasingly support subjective data collection. The most common way to collect subjective data is to use experience sampling, that is, explicitly ask for user feedback at regular intervals or in specific situations.[1] One alternative is to automatically infer meaningful descriptions from sensor data.

Several researchers have developed various platforms that support objective data collection. Most of these platforms are limited to a specific runtime environment or to a specific set of sensors. For example, ContextPhone logs various phone events (phone and application usage, for example), but can be used only on Nokia S60 devices.[2] Platforms that support multiple runtime environments are typically limited to a specific set of sensors or data type. For example, Intel PlaceLab[3] is limited to location data and BeTelGeuse's earlier version supports only Bluetooth-enabled sensors.[4]

Several platforms that support objective and subjective data collection have been proposed. Most of these platforms only run on devices from the Microsoft Windows CE operating system family. The first such tool was the Context-Aware Experience Sampling tool (CAES), which runs on PDAs using the Microsoft PocketPC operating system.[5] However, the CAES tool is no longer supported (the project was last updated in 2003). The most comprehensive platform so far is MyExperience,[6] which supports a wide range of sensors, contains a comprehensive event mechanism, supports a variety of experience sampling modalities, and has been extensively tested. Unfortunately, MyExperience is restricted to mobile devices running the Windows Mobile operating system.

Frameworks that automatically infer higher-level contexts from sensor data have been proposed. These systems typically focus on a specific contextual variable, and they don't have generic data collection capabilities. Examples include Opportunity Knocks,[7] which focuses on location information, and the Context Recognition Network, which focuses on activity information.[8] Because these frameworks focus only on the detection of activities, they serve the same purpose as the plug-ins in BeTelGeuse.

BeTelGeuse's main advantages are its support for multiple platforms and that its sensing capabilities scale according to the client device's capabilities. Thus, researchers can use BeTelGeuse on most platforms, but the amount and nature of data collections depends on the target device's available sensors and capabilities. Additionally, BeTelGeuse isn't limited to merely logging data; it can automatically and nonintrusively infer higher-level context from sensor data. Although earlier platforms support (subsets of) these features, BeTelGeuse is the first platform to support all of them. One of BeTelGeuse's limitations is that it currently doesn't contain built-in experience sampling functionality, but we're working on a plug-in for that.

REFERENCES

1. S. Consolvo and M. Walker, "Using the Experience Sampling Method to Evaluate Ubicomp Applications," *IEEE Pervasive Computing*, vol. 2, no. 2, 2003, pp. 24–31.
2. M. Raento et al., "ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications," *IEEE Pervasive Computing*, vol. 4, no. 2, 2005, pp. 51–59.
3. T. Sohn et al., "Experiences with Place Lab: An Open Source Toolkit for Location-Aware Computing," *Proc. 28th Int'l Conf. Software Engineering* (ICSE 06), ACM Press, 2006, pp. 462–471.
4. P. Nurmi et al., "BeTelGeuse—A Tool for Bluetooth Data Gathering," *Proc. 2nd Int'l Conf. Body Area Networks* (BodyNets), ACM Press, 2007; http://portal.acm.org/citation.cfm?id=1460232.1460253&coll=GUIDE&dl=GUIDE&CFID=25695462&CFTOKEN=85841578//.
5. S.S. Intille et al., "A Context-Aware Experience Sampling Tool," *CHI 03 Extended Abstracts on Human Factors in Computing Systems*, ACM Press, 2003, pp. 972–973.
6. J. Froehlich et al., "MyExperience: A system for in situ Tracing and Capturing of User Feedback on Mobile Phones," *Proc. 5th Int'l Conf. Mobile Systems, Applications and Services* (MobiSys), ACM Press, 2007, pp. 57–70.
7. D.J. Patterson et al., "Opportunity Knocks: A System to Provide Cognitive Assistance with Transportation Services," *Proc. 6th Int'l Conf. Ubiquitous Computing* (UbiComp 04), LNCS, Springer, vol. 3205, 2004, pp. 433–450.
8. D. Bannach et al., "Distributed Modular Toolbox for Multi-Modal Context Recognition," *Proc. 19th Int'l Conf. Architecture of Computing Systems* (ARCS 06), vol. 3894, LNCS, Springer, 2006, pp. 99–113.

- *High-level context*. Existing platforms are limited to gathering raw sensor measurements rather than inferring high-level abstractions of the user's location. High-level abstractions are often more meaningful and provide better clues about the user's actual situation, motivation, and information needs.

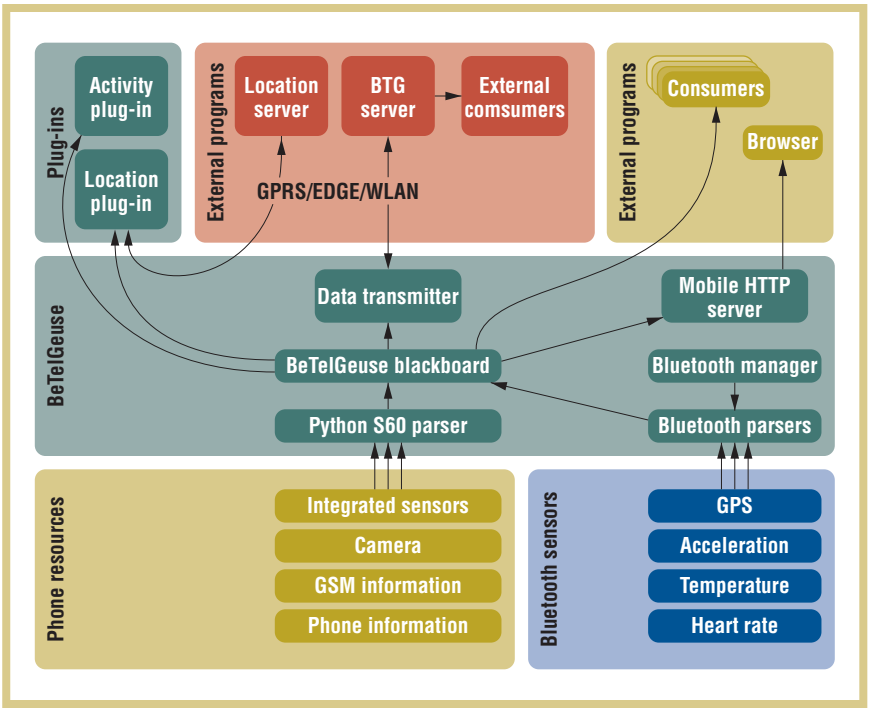

Figure 1. BeTelGeuse's high-level architecture. BeTelGeuse follows the microkernel architecture pattern.

- *User experience.* BeTelGeuse is aimed at two interconnected user groups: researchers who run user studies and extend BeTelGeuse by writing custom parsers or plug-ins, and users or study participants who run BeTelGeuse on their personal devices as part of a study or for their personal use (GPS traces or heart-rate data, for instance). From a researcher's perspective, user experience implies that BeTelGeuse should be easy to configure. For study participants, user experience refers to the platform's generic usability and that the tool doesn't have a noticeable impact on the client device's performance.

We describe BeTelGeuse's architecture in more detail.

BeTelGeuse Architecture

BeTelGeuse's high-level system structure is inspired by the microkernel architecture pattern. We have a separate core that offers the smallest set of functionality needed to run the tool. The core also defines interfaces for components that provide extended functionality. This allows a single implementation of the main functionality and custom extensions for different runtime environments. Our design has also been inspired by other context-aware frameworks. Similar to widgets in the Context Toolkit,[3] parser components act as abstractions of sensors, and we use a blackboard architecture inspired by the work of Terry Winograd.[4] Figure 1 shows BeTelGeuse's high-level architecture. The BeTelGeuse core contains data gathering, Bluetooth discovery, parser interfaces, the blackboard, and data transmission classes.

Implementation

To fulfill our goal of multiplatform support, we implemented BeTelGeuse's core using Java Micro Edition. We only use features that are commonly available in the Java implementations of most mobile and desktop devices. More specifically, BeTelGeuse's core is compatible with mobile systems that conform to the Mobile Information Device Profile (MIDP) 2.0 (http://jcp.org/aboutJava/communityprocess/final/jsr118) and Connected Limited Device Configuration (CLDC) 1.1 (http://jcp.org/aboutJava/communityprocess/final/jsr139) specifications, which are based on Java 1.3 and make BeTelGeuse compatible with desktop systems. Additionally, BeTelGeuse requires a JSR-82-compliant (http://jcp.org/aboutJava/communityprocess/final/jsr082) Java Bluetooth stack.

These devices satisfy BeTelGeuse's platform requirements:

- Mobile phones that support Java and Bluetooth, such as second and third editions of various Nokia S60 devices and Sony Ericsson W800i devices. (For a list of more than 100 compatible devices, please see http://developers.sun.com/mobility/device/pub/device/list.do).
- GNU/Linux PCs that run a standard Java installation with the freely available AvetanaBluetooth Java Bluetooth stack installed (http://sourceforge.net/projects/avetanabt).
- Windows PCs that run a standard Java installation and the freely available Bluesock Java Bluetooth stack (https://bluesock.dev.java.net).
- PDAs with Microsoft Windows Mobile running IBM J9 Java Virtual Maching with a commercial version of AvetanaBluetooth. We've tested BeTelGeuse on a Hewlett Packard hx4700 PDA running Microsoft Windows Mobile 2003 2nd Edition.

On smartphones, MIDP-specific socket connection classes are plugged into the core. For GNU/Linux and Windows, we used Java 1.5 socket classes. The Bluetooth parsers remain the same across platforms. We've included platform-specific parsers depending on the device that BeTelGeuse is deployed on—we enabled Python S60 extensions, for example, on Nokia S60 smartphones. BeTelGeuse's extensibil-

TABLE 1
Sensors currently supported by BeTelGeuse.

Sensor	Examples of measured data	Examples of events
Bluetooth GPS (NMEA 0183) sensors	latitude, longitude, altitude, time, number of satellites	LatitudeChange, LongitudeChange, timestamp, value equal/greater/smaller than a specified threshold
Alivetec Heart Monitor (www.alivetec.com)	ECG, 3-axis acceleration	ecgChange, accelerationChange, value equal/greater/smaller than a specified threshold
I-CubeX (http://infusionsystems.com)	distance (ultrasound), 3-axis acceleration, temperature, humidity, orientation, background light	value change events, value equal/greater/smaller than a specified threshold
Python S60 Parser (Nokia S60 3rd edition phones, requires signed Python)	GSM cell information: identifier, area, network and country codes, network name, signal strength. Call and SMS data: outbound and inbound calls and SMS, SMS access times. Phone status information: battery level, phone profile	value change events, value equal/greater/smaller than a specified threshold, callStart, callEnd, callAnswer, msReceive, smsOpen, smsSend, profileChanged, batteryLow
Local device	Bluetooth proximity information (In periodic scanning mode)	deviceAddressPresent, deviceNamePresent, deviceLost
Core	BeTelGeuse internal events source	parserCreated, parserDisconnected, parserDeviceLost, parserModeStreaming, parserModeRequest

ity also makes it possible to integrate platform-specific tools, such as MyExperience or ContextPhone.

Configuration

BeTelGeuse loads parameter values from a configuration file on startup. The configuration specifies Bluetooth mappings, frequency of data polling on each sensor, and whether to send data to a server or save it on the device. Users can modify the configuration through a MIDlet user interface. Alternatively, researchers or the study participants can specify a custom configuration file. We're currently implementing support for modifying the configuration remotely via command messages to the BeTelGeuse blackboard.

Blackboard

The BeTelGeuse blackboard acts as a hub for communications between different components and lets external components access the blackboard, such as when providing or receiving context data. BeTelGeuse Java plug-ins connect to the blackboard using direct method calls, whereas external components and plug-ins must use a local socket.

The blackboard uses a Simple Sensor Interface-like protocol (SSI; www.ssi-protocol.net). The messages begin with a command code, and most have component-type, user-id, and component-id following the code. The blackboard confirms command messages, but not data packets. The command code identifies the message. For example, "c" identifies a create message, which results in the blackboard creating a receiver and data container for the caller. The component-type, component-id, and user-id specify the message's target components (a subset or all of the components). This is useful in scenarios in which a number of components want to establish a dialogue. The current protocol version lets external components create, delete, and list components and components' owners, and download or upload data.

Data on the blackboard resides in memory. The blackboard is data-type agnostic and views the data as an opaque string of bytes. Components reading the data are responsible for interpreting it. By default, blackboard components interact in a publish–subscribe communication pattern. When a component receives new data, it notifies the blackboard, which, in turn, notifies consumers, that is, other components that have subscribed to the data. Each new data packet overwrites previous field values of the same component (GPS coordinates override old ones, but a longitude value only overwrites the old longitude value). Components might subscribe to receive data whenever a specific event occurs. For example, the GPS can be read at regular intervals or whenever the GSM cell changes. If a component subscribes to data changes, it's only notified when the data changes in the specified magnitude. Table 1 lists other supported event types. Components aren't required to subscribe to events; for example, parsers produce data but don't consume data produced by other blackboard components.

Data Transmitter

Although programs on the client device can access context data via the blackboard, remote researchers or external applications can't access data this way. To achieve data accessibility, BeTelGeuse contains a data transmitter, which synchronizes data with remote or local persistent storage and

makes it available to external components. Web applications can access data using the mobile HTTP server.

We implemented remote storage using a server-side component that stores the context data into a MySQL database. The data transmitter sends data using any Internet connectivity method that the client device supports—3G, GPRS, or wireless LAN, for example. The communications use a lightweight protocol, implemented on top of TCP. The protocol borrows ideas from existing sensor protocols, especially the SSI protocol, which is well-suited for communications between sensors and a controlling device. However, the SSI protocol is insufficient for our purposes because it doesn't contain messages for sending sensor names and identifiers, sending incremental sensor information, establishing a persistent session, or reconnecting to an existing session.

When local storage is used, data is stored on the device in a sequential file. The file resembles a data transmission log and the data transmitter can upload it to the BeTelGeuse server when Internet connectivity is available. Currently we don't support automatic replay of the transmission log, but the file can be uploaded manually. Internet connectivity rapidly drains the client device's battery life, but modern mobile devices support memory cards with a capacity of several gigabytes, so we can store several months of data locally.

Bluetooth Manager

The Bluetooth manager scans for Bluetooth devices and manages connections to Bluetooth sensors. The scanning can be performed periodically or initiated manually. Users can configure the scan interval using the MIDlet user interface or through remote access. Scanning in periodic mode is advantageous because it enables collecting (Bluetooth) proximity data, such as for social network analysis.[5] The periodic mode facilitates sensor management with stationary sensors scattered around the environment. In manual mode, Bluetooth scanning is performed at startup, after which users must manually trigger the scans using the MIDlet interface. This mode is useful when the sensor configuration doesn't change and proximity data isn't needed. Manual mode also helps avoid Bluetooth usage conflicts between scans and sensor data transmissions. On certain devices, such as older Nokia S60 second edition smartphones, Bluetooth scans require exclusive access to the Bluetooth stack, which can cause the receiving Bluetooth buffer to overflow with sensor data. Manual mode also slightly improves battery life.

The Bluetooth manager automatically connects to devices that users specify in the configuration and instantiates appropriate parsers. A device is specified by its (partial) friendly name (contains "GPS," for example) or Bluetooth address. Users can add new devices using the MIDlet interface or by editing the configuration file. When a Bluetooth scan finishes, the Bluetooth manager connects any matching discovered devices and creates appropriate parsers.

Connections to sensor devices might be lost for various reasons: wireless communication might be blocked, nearby devices can cause interference, sensor batteries might fail, or the sensors might become unreachable as users move. When a sensor connection is lost, the Bluetooth manager tries to restore the connection. The reconnection mechanism is based on an exponential back-off scheme. After a user-configurable maximum timeout is reached, the Bluetooth manager drops the sensor or continues the reconnection attempts with the maximum timeout, depending on the configuration.

Mobile HTTP Server

Because Web applications are increasingly based on locally executed JavaScript, we can easily enable Web applications to access context data. Our solution integrates a lightweight component, which reads HTTP requests and returns context data, into the BeTelGeuse core. Web applications that run on the device's browser can access context data using Ajax. We support `HEAD`, `GET`, and `POST` requests that follow the HTTP 1.0 specification. To minimize overhead and delays, the server simply reads the URL and query string from the HTTP request (/index.html?param=value) and returns sensor data. By default, the mobile HTTP server returns the context information in JavaScript Object Notation (JSON), which makes the information directly accessible as native objects and data structures for JavaScript code running on the device's Web browser. The mobile HTTP server also supports HTML and text formats. By default, all context data is returned, but the server can also be queried for a specific sensor's data. We use a query mechanism based on a Unix-style directory format. The URL, http://localhost/gps/latitude, for example, returns only GPS latitudes. Web applications can specify the data format separately as a query parameter: the call http://localhost/gps/latitude?format=html returns the latitude in HTML. Similar to the data transmitter and the BeTelGeuse server, the mobile HTTP server facilitates data access and contributes to our data accessibility goal.

Because Web applications are increasingly based on locally executed JavaScript, we can easily enable Web applications to access context data.

Context Parsers

Context parsers are abstractions of sensors that read and parse data, and write it to the blackboard. The parsers can operate in streaming or request mode. In streaming mode, data is continuously read from the sensor, whereas, in request mode, the sensor is polled for data when a certain event occurs or at user-configurable intervals. The request mode is useful for long-term data collection.

BeTelGeuse-compatible sensor types that can be used include external Bluetooth sensors, integrated phone sensors, software sensors, and Internet sensors. An Internet sensor could read Google Calendar entries, for example, and push the data to the blackboard. Developers can limit sensors to a specific platform. We use a platform-specific parser on Nokia S60 devices, for instance. Developers can also integrate BeTelGeuse with sensors using another communication technology. Table 1 lists the sensors BeTelGeuse currently supports.

To add support for new sensor types, developers must implement a new context parser or extend an existing one. Developers might also implement context parsers, written primarily in Java, as external plug-ins that push their data to the blackboard. When writing a parser for a new Bluetooth sensor, developers must assign the parser with an identifier and register it with the Bluetooth manager to ensure that the Bluetooth manager can automatically instantiate the parser. The BeTelGeuse Web site has details about implementing a parser and registering it with the Bluetooth manager.

Plug-ins and Extensions

Researchers can extend BeTelGeuse in various ways. They can develop new parsers or plug-ins. Additionally, they can develop Web-based extensions on top of the data transmitter.

Python S60 Parser

The Python S60 Parser provides information about phone status and usage on Nokia S60 devices and access to internal phone sensors. Currently, BeTelGeuse supports GSM information, call and SMS logging, and phone status information. The Python S60 Parser works similarly as other parsers. Thus, it registers to the blackboard, and other components can subscribe to data or events from the parser. In addition, it's possible to access phone calendar information or internal sensors, such as the microphone, camera, or integrated GPS.

Location Plug-in

The location plug-in consists of a server module and two client-side modules: a GSM positioning module and SerPens.[6] The GSM positioning module uses GSM fingerprinting to estimate the users' location whenever GPS is unavailable. SerPens is a collaborative tagging tool that lets users assign public and private labels to their locations. The labels are tied to a taxonomy that supports different granularities, such as country, region, and street. Users use private tags to indicate personally meaningful locations whereas public tags can be used to label landmarks. SerPens provides more meaningful data than mere coordinates, and can give clues about the context of the user. The location plug-in helps to achieve our high-level context information goal.

Researchers can develop new parsers, plug-ins, or Web-based extensions for BeTelGeuse.

Activity Plug-in

Whereas location typically provides clues about users' generic situation (home, work, school, and so on), activity information can provide clues about the environmental constraints that influence users' interaction. For example, while users are walking, they must pay attention to the environment, which often results in abrupt bursts and short-lived interaction patterns. Thus, mobile user studies could significantly benefit from detailed activity information. BeTelGeuse contains an activity plug-in that detects basic activities from accelerometer data. Our current implementation supports the Alivetec Heart Monitor (see Table 1) and detects the following activities: resting, walking, brisk walking, jogging, running, and commuting. We're currently modifying our plug-in to support the built-in accelerometers of recent Nokia smart phones and extending the number of identified activities.

BeTelGeuse Server

External applications or remote researchers running a multiperson study often need easy access to data from several devices. To facilitate data access, we've implemented a server component that maintains sessions with connected BeTelGeuse instances. It receives data sent by each instance and stores it in a MySQL database. The BeTelGeuse server uses a series of data filters for distributing data to external programs on reception. For example, the location server uses the BeTelGeuse server to obtain relevant location data (GSM + GPS).

Performance Evaluation

In analyzing BeTelGeuse's impact on client devices, we consider two aspects of performance: the memory requirements and the impact on battery life.

BeTelGeuse's memory footprint is between 5.6 and 7.3 Mbytes, depending on the configuration and the amount of sensors that are connected. The local version requires somewhat less memory than the online version. These figures include the memory required to run the Java virtual machine. In terms of instal-

TABLE 2
Battery lifetime in hours under different configurations.

#	Experiment setup	Mean	Standard deviation
1	Python	35.6	0.34
2	Python, Periodic GPS	34.1	0.44
3	Python, GPS Streaming	31.5	0.48
4	Python, GPS Streaming, BT Scan	25.0	0.24
5	Python, Server	6.0	0.15
6	Python, Server, Periodic GPS	5.8	0.28
7	Python, Server, BT Scan	6.2	0.05
8	Python, Server, Periodic GPS, BT Scan	5.8	0.54
9	Python, Server, GPS Streaming, BT Scan	5.0	1.04

lation size, BeTelGeuse requires only 179 Kbytes.

To measure the impact on the client device's battery life, we conducted a set of experiments in which we used BeTelGeuse under different configurations and measured the time it took to drain the battery of five fully-charged, brand-new Nokia E61i devices (with standard BP-4L 1500 mAh batteries). We considered nine different configurations and averaged the results over the devices. As we considered new devices, our results should be interpreted as upper bounds for performance. However, the homogeneity of the devices lets us draw better conclusions about the performance differences.

Table 2 shows our results. As our baseline, we used a version in which only the Python S60 parser collects data. This version lasted between 35 and 36 hours. Adding a GPS device that was read once per minute decreased the battery lifetime to 34 hours. Running Bluetooth scans on top of this had only a minor impact. These values span well over a day, which makes these setups well suited for long-term data collection. Changing the GPS from periodic reading mode to continuous streaming had a more significant effect on the battery lifetime, with the mean lifetime decreasing to 25.7 hours. Again, the Bluetooth scanning had only a minor impact on the performance (mean 25 hours). Thus, BeTelGeuse's battery usage is well-optimized with respect to Bluetooth.

The final experiments measured the effect of Internet connectivity on battery usage. In these experiments, we configured the transmitter to send data once every minute. As Table 2 indicates, the mean lifetime is roughly 5 to 6 hours, with the variation being caused by the amount of Bluetooth connectivity. For long-term data collection, the transmission rate should be decreased.

Case Studies

We used BeTelGeuse to collect large amounts of context data, and integrated it as a context source into a mobile application.

Gathering and Analyzing Location Data

We've used BeTelGeuse to collect GSM and GPS data from seven users for more than one month. The participants used a Nokia E61i mobile phone and an external Bluetooth GPS receiver. The GPS was polled every 60 seconds. We also scanned for nearby Bluetooth devices and gathered internal phone information with the Python S60 parser. An informal user study indicated that BeTelGeuse was easy to use, but participants considered the GPS receiver's short battery lifetime inconvenient. We've also used BeTelGeuse to collect GPS traces with different spatial and temporal characteristics from 10 to 15 different countries.[2]

Context-Adaptive Widgets

Capricorn is an adaptive, Web-based widget engine for mobile devices,[7] which uses BeTelGeuse as a context source and enables widgets to adapt their information based on the user context. For example, a context-aware travel planner can automatically fill in the origin of travel using location information provided by SerPens, and a news or weather service can provide localized information using location information provided by BeTelGeuse. As Capricorn is a Web application, it accesses context information through the BeTelGeuse mobile HTTP server on the device.

We're currently improving the data transmitter by adding support for data encryption and event-based connectivity. We're also extending BeTelGeuse to use new context sensors and continuing to develop additional plug-ins.

the AUTHORS

Joonas Kukkonen is a research assistant at the Helsinki Institute for Information Technology HIIT. His research interests include recommender systems and personalization of mobile applications. He is an MSc student in the Department of Computer Science at the University of Helsinki. Contact him at joonas.kukkonen@cs.helsinki.fi.

Eemil Lagerspetz is a research assistant at the Helsinki Institute for Information Technology HIIT and an MSc student at the University of Helsinki. His research interests include mobile data management and data communications. He has a BSc in computer science from the University of Helsinki. Contact him at eemil.lagerspetz@cs.helsinki.fi.

Petteri Nurmi is a researcher at the Helsinki Institute for Information Technology HIIT. His research interests include personalization of mobile applications and services, and mobile human computer interaction. He has an MSc in computer science from the University of Helsinki. Contact him at petteri.nurmi@cs.helsinki.fi.

Mikael Andersson is a research assistant at the Helsinki Institute for Information Technology HIIT and an MSc student in communications engineering at the Helsinki University of Technology. His research interests include computer networking with a focus on application layer services, and locationing and data-gathering on mobile devices. Contact him at mikael.andersson@tkk.fi.

Specifically, we're extending the activity recognition plug-in and developing an experience sampling plug-in that supports context-triggered questionnaires. Furthermore, we're planning to use BeTelGeuse in various context-aware applications as a context source.

REFERENCES

1. J. Lester, T. Choudhury, and G. Borriello, "A Practical Approach to Recognizing Physical Activities," *Proc. 4th Int'l Conf. Pervasive Computing* (Pervasive 06), vol. 3968, LNCS, Springer, 2006, pp. 1–16.
2. P. Nurmi and S. Bhattacharya, "Identifying Meaningful Places: The Nonparametric Way," *Proc. 6th Int'l Conf. Pervasive Computing* (Pervasive 08), vol. 5013, LNCS, Springer, 2008, pp. 111–127.
3. A.K. Dey, G.D. Abowd, and D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human-Computer Interaction*, vol. 16, nos. 2–4, 2001, pp. 97–166.
4. T. Winograd, "Architectures for Context," *Human–Computer Interaction*, vol. 16, no. 2, 2001, pp. 401–419.
5. N. Eagle and A.S. Pentland, "Reality Mining: Sensing Complex Social Systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, 2006, pp. 255–268.
6. S. Bhattacharya et al., "SerPens—A Tool for Semantically Enriched Location Information on Personal Devices," *Proc. 3rd Int'l Conf. Body Area Networks*, ICST, 2008; http://portal.acm.org/citation.cfm?id=1460257.1460297&coll=GUIDE&dl=GUIDE&CFID=25695462&CFTOKEN=85841578//.
7. F. Boström et al., "Capricorn—An Intelligent User Interface for Mobile Widgets," *Proc. 10th Int'l Conf. Human-Computer Interaction* (MobileHCI 08), ACM Press, 2008, pp. 328–330.