6

Applications

This chapter describes the applications that have been built using the GAMBAS middleware. To do this, the chapter briefly outlines the integration of the system components described in Chapter 3, Chapter 4 and Chapter 5. Based on this description, it introduces the application development support provided by GAMBAS for different execution environments. To clarify this, we present a number of simple but full-featured applications that leverage the different components of the middleware. Based on this, we then describe the two large-scale applications that have been built with the middleware. These applications focus on realizing significant parts of the mobility scenario and the environmental scenario introduced in Chapter 1 that motivated the work on the GAMBAS middleware.

6.1 Application Development Support

In the following, we describe how the GAMBAS middleware is used during application development. To do this, we first briefly review how the different middleware components described in the previous chapters are integrated into a single system. Thereafter, we discuss how different execution environments are supported through the GAMBAS SDK and middleware runtime. Finally, we present a number of simple applications that have been built with the SDK to demonstrate the different features offered by GAMBAS.

As shown in Figure 6.1, the integrated GAMBAS system consists of (1) a number of networked devices executing the GAMBAS middleware and (2) the GAMBAS Dynamic Data Registry. Each device may execute one or more GAMBAS applications (or simply apps) using the GAMBAS middleware. An example for such an app is an Android application executed by an end user on his smart phone. Another example would be server software

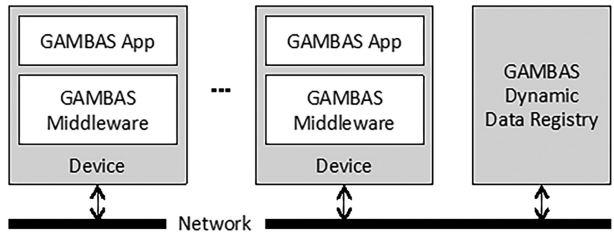


Figure 6.1 Integrated System.

executed by a service provider on a dedicated server connected to the Internet. As described in Chapter 4, the GAMBAS Dynamic Data Registry is a generic service that provides devices with the ability to discover data sources. The functionality of this registry is comparable to the Domain Name System (DNS), which provides name resolution on the Internet. Although GAMBAS assumes that this functionality is provided publicly, GAMBAS allows developers to run their own registry during development and testing. Since the functionality of the registry has been described in detail in Chapter 4, in the following, we focus on the remaining functionalities.

6.1.1 Overview

Figure 6.2 gives an abstract overview of the middleware structure. The integration is realized by: (1) a set of interfaces, support libraries and tools called the *Software Development Kit* (SDK) and (2) the *GAMBAS CoreService* which provides the accompanying runtime environment. The Software Development Kit (SDK) in turn consists of two parts. The *Service Programming Interface* (SPI) is used to develop GAMBAS functionality and integrate it into the middleware. The *Application Programming Interface* (API) is used to develop GAMBAS apps. The CoreService sets up the GAMBAS middleware and manages the life cycle of GAMBAS system components. Each system component encapsulates the implementation of one of the core GAMBAS parts described in Chapter 3, Chapter 4 and Chapter 5, e.g. the Semantic Data Storage (SDS) or the Data Acquisition Framework (DQF).

In addition, the CoreService integrates a special communication system component that encapsulates an extended version of the BASE communication middleware discussed in Chapter 5. This enhances GAMBAS with communication support to interact with remote GAMBAS devices. Furthermore, the CoreService realizes the SPI by linking each system component to all other components that they use during their own execution via interfaces from

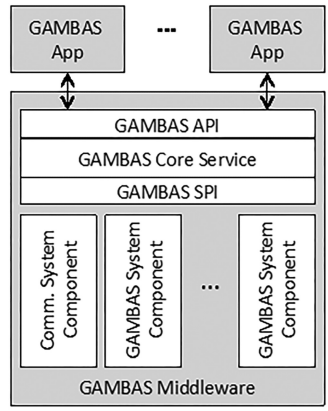


Figure 6.2 Abstract Middleware Structure.

the SPI. This effectively provides a tight and efficient integration between the components without inducing dependencies to their actual implementation. Finally, the CoreService implements the GAMBAS API towards GAMBAS applications in both Android and J2SE environments. To do this, it receives calls, forwards them to the right system component and delivers results back to the original caller.

Due to the intrinsic differenced between Android and J2SE execution environments, the abstract structure shown in Figure 6.2 has two distinct concrete implementations. In the following, we briefly describe their differences and similarities.

6.1.2 J2SE Support

GAMBAS for J2SE specializes and implements the generic middleware architecture described before for server systems running J2SE. This allows service providers to integrate their services into the GAMBAS platform. Figure 6.3 shows the resulting system architecture. Since the J2SE version of GAMBAS is primarily intended for the development of server systems, it does not include support for user interfaces. Clearly, service providers will, in many cases, add their own user interface, e.g. based on web technologies. This, however, is outside of the scope of GAMBAS and thus not explicitly supported. All other GAMBAS system components mentioned previously are integrated, namely communication, data acquisition (the DQF), data storage (the SDS) and querying (the xQP), as well as security and privacy (the PRF).

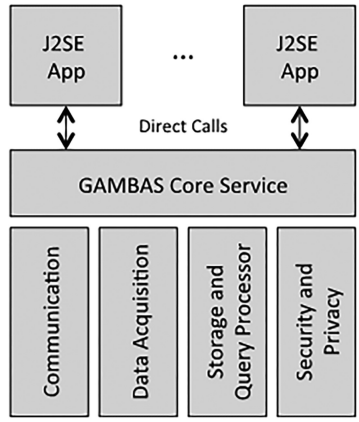


Figure 6.3 GAMBAS for J2SE.

As described before, the CoreService realizes the SDK and manages the life cycle of the whole GAMBAS system and all its components on a local device. GAMBAS for J2SE is implemented as a library that is linked to an application using it. To start using the GAMBAS system, an application has to first import and instantiate the CoreService. The CoreService can be configured by passing it an instance of *CoreSetting*. This allows the application to specify, e.g. the address of the GAMBAS data registry and communication gateway as well as a pseudonym that should be used to address the system. Settings can be changed dynamically and the CoreService will perform any necessary updates automatically, e.g. when a pseudonym should be changed. When the CoreService is instantiated (and thus started) it instantiates, configures and starts in turn all necessary GAMBAS system components.

To decouple life cycle management of components from their actual implementation, each component is encapsulated by a specific subclass of *AbstractSystem*, providing, e.g. methods for startup and shutdown. As an example, the SDS is integrated by sublassing AbstractSystem with a new class *DataStorageSystem*, which implements all life cycle management functions independently of the actual SDS implementation. This way, the SDS is independent of the CoreService and can, e.g. be reused in other contexts without other GAMBAS components. The CoreService also passes each component references to all other components it may require. As an example, the query processor uses the data storage, the communication system and the privacy manager. The CoreService enables this by passing references to these

three components to the query processor. At runtime, the query processor calls these components directly, without using the CoreService anymore.

To use functionality of the GAMBAS middleware, e.g. to store data, applications can call a number of methods on the CoreService. The CoreService in turn forwards this request to the corresponding local system component, retrieves results from it and forwards them to the calling application. This design was chosen over directly exposing system components to applications because it allows the CoreService to impose further checks on the correctness, security and consistency of these calls, if needed. As an example, the CoreService might deny a new request if it has already started shutting down the system. In addition to this, this approach also provides access transparency for system components, i.e. it allows us to decouple the system components from the way they are called. If an application wants to call a remote system component (e.g. to store data in a remote SDS), it can do so by calling a local method on the CoreService. The CoreService will forward this request to the communication system (essentially acting as a communication broker), which will send it to the remote system. There, the incoming request will be forwarded by the communication system to the CoreService, which in turn forwards it to the corresponding system component. Finally, if an application wants to stop the GAMBAS middleware, it again calls the CoreService, which notifies all components and shuts down the system correctly.

As a result, the CoreService is the central component of this architecture. It is responsible for receiving and answering all local and remote calls from applications, mediates all dependencies between system components and fully manages their life cycles. This encapsulates nearly all integration activities into it, reduces the complexity of implementing the actual system components and allows them to focus on their core functionality.

6.1.3 Android Support

In addition to J2SE devices, the GAMBAS middleware also directly supports application development on Android devices. From a high-level perspective, the Android integration is similar to the J2SE version, but when looking at the details, it has two main differences: first, it separates the core middleware from applications using it, reflecting the distinct Android runtime model and reducing the overall resource need of the system. Second, it includes additional support for user interactions with the intentional user interface (IUI). The resulting architecture can be seen in Figure 6.4.

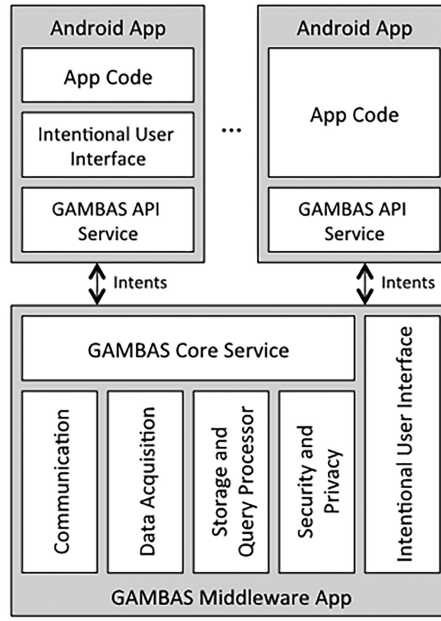


Figure 6.4 GAMBAS for Android.

6.1.3.1 GAMBAS Middleware App

On Android, the main functions of the GAMBAS middleware are realized as a stand-alone Android app instead of a linkable library. This app is independent of any third-party Android applications using it. On Android, the life cycle of an app is controlled by the OS. It may at any time pause or stop/destroy any app, if it requires more resources. If the GAMBAS middleware would be linked to an app using it, the OS could decide to stop it, if the app is not used by the user right now. By separating the middleware into its own app, we are separating its life cycle management from that of all apps that use it. In addition, this design allows us to efficiently share a single instance of the middleware between all third-party apps, reducing the needed resources and thus allowing the OS to keep all apps active in memory for a longer time.

An alternative approach would be to model the middleware as an Android service. However, allowing the middleware to have its own user interface – independently of any other app – allows us to integrate all configuration activities that a user wants to perform for the whole system in one place. In addition, it also allows the user to start and stop the middleware explicitly,

since its execution will reduce battery lifetime. To remind the user that GAMBAS is running, we display a corresponding icon in the Android status bar. By clicking this icon, the user can display the middleware user interface and control its behavior, e.g. reconfigure or stop it.

The separation of the middleware into a distinct app also influences how third-party apps can access its functionality. Direct calls are no longer possible since the apps are running in separate processes. Therefore, we use Android intents to interact with the middleware. Intents are small events or messages that a process can publish and that can be received by other processes. To use the GAMBAS middleware, an application can publish a number of intents that are received by the middleware. The CoreService includes support for this. It translates the intents into direct calls and forwards them to the corresponding GAMBAS system components. Once a result is available, the CoreService translates it back into an intent and publishes it, allowing the original app to receive it.

Clearly, this is more complicated for app developers than directly calling a method on a Java object. To reduce the complexity of the interface, we provide a GAMBAS API service. This service is realized as a Java class that can be subclassed by an application developer. It already includes all necessary functionalities to translate calls to the middleware into intents and vice versa as well as additional support for handling life cycle and error. Thus, by using this API service, the app developer can access the middleware without knowing about the specifics of Android interprocess communication.

6.1.3.2 GAMBAS User Interface

As described above, the GAMBAS middleware for Android devices contains the Intent Aware User Interface (IUI). The IUI is separated into two parts: an interface to control the behavior of the GAMBAS middleware itself and support for the development of user interfaces of third-party apps.

The GAMBAS middleware user interface enables the user to configure a multitude of aspects (Figure 6.5(b)) such as the middleware life cycle (Figure 6.5(a)), the used data discovery registry and communication gateway (Figure 6.5(f)), the user's pseudonym, known friends, their keys (Figure 6.5(d)) and privacy policies. This allows users to inspect and adapt the current system state in one integrated place and makes it much easier for them to understand what data is currently made available to whom.

In addition, the middleware user interface allows to manage all third-party GAMBAS apps (Figure 6.5(c)) in an integrated view. This view allows to

Figure 6.5 User Interface. (a) Start, (b) Settings, (c) Apps, (d) Privacy, (e) Features, (f) Development.

install new apps from the Google Market, to start them and to remove them once they are no longer needed. Finally, in order to give full control over sensing to the users, the user interface also enables users to disable the different data collection components offered by the data acquisition framework.

6.1.4 Application Examples

To test and showcase the GAMBAS SDKs, we have developed several applications that demonstrate the use of the different features of the GAMBAS middleware. These application have been made available to developers and they have also been published on the Android market. In the following, we briefly describe three of these applications. We first describe the application functionality and then map it to the middleware functionality.

6.1.4.1 GAMBAS Voiceprint Launcher

The GAMBAS Voiceprint Launcher is an Android application developed on top of the GAMBAS middleware. The application uses the voiceprint technology developed as part of the data acquisition framework (c.f. Chapter 3). The Voiceprint Launcher enables a user to launch an application by issuing a voice command. To enable the launching of applications via a voice command, the user first needs to train the launcher by creating recordings of the commands that shall start different applications (see Figure 6.6).

To do this, the user can add an application from the list of applications installed on the device. Then, the user can select the application and press the train button (i.e. the button with the white headset) to start the training. Alternatively, the user can press the delete button (i.e. the button with the white trash can) to delete the application and all training data. Once the

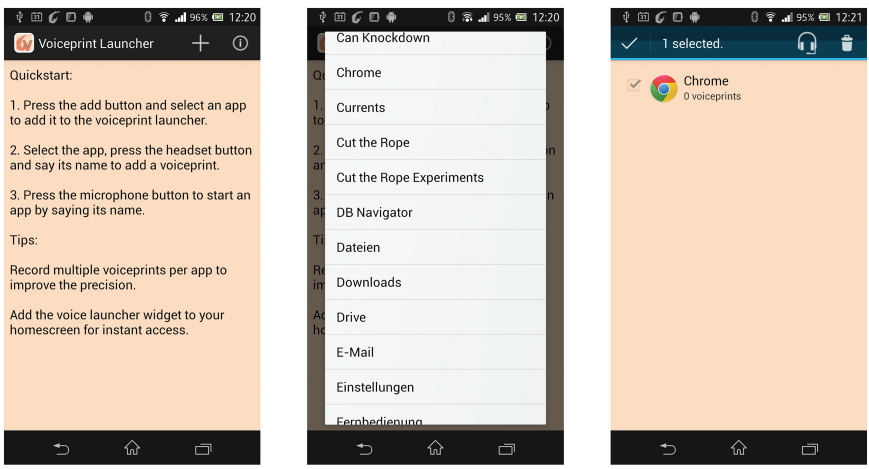


Figure 6.6 Voiceprint Luncher Training.

user has pressed the train button, a dialog appears that prompts the user to say the application name loud. Once the user completed this, the application computes a voiceprint and stores it locally.

As soon as the user has trained one or more applications, he can start the application by pressing the start button (i.e. the white microphone). This will open up a dialog that prompts him to say the application name out loud. Once he has done that, the application will compute a voiceprint and match it against all stored voiceprints. The closest match will be selected and the associated application will be started. Alternatively, the user can also add the voiceprint launcher widget to the home screen of the device. This allows the user to directly access the launcher (see Figure 6.7).

From a technical perspective, the GAMBAS Voiceprint Launcher demonstrates a substantial part of the middleware. However, it is noteworthy that it solely executes locally on the phone of a user and thus, it does not require any remote connectivity or services. Consequently, it does not cover any communication-related aspects and it also does not cover the J2SE integration. As depicted in Figure 6.8, the GAMBAS Voiceprint Launcher extensively uses the GAMBAS middleware on Android through the Android SDK that connects it with the core service of the GAMBAS Middleware App.

Of the functionality provided by the core service, the GAMBAS Voiceprint Launcher uses four out of five building blocks as follows:

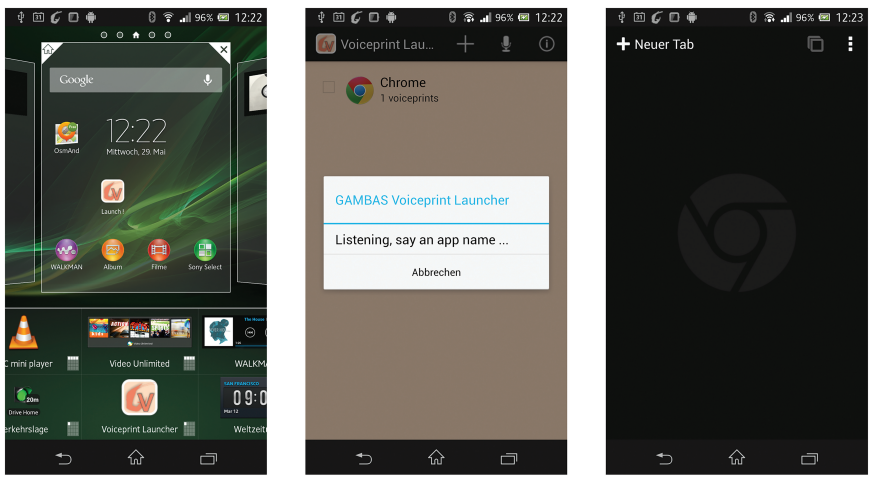


Figure 6.7 Voiceprint Launcher Usage.

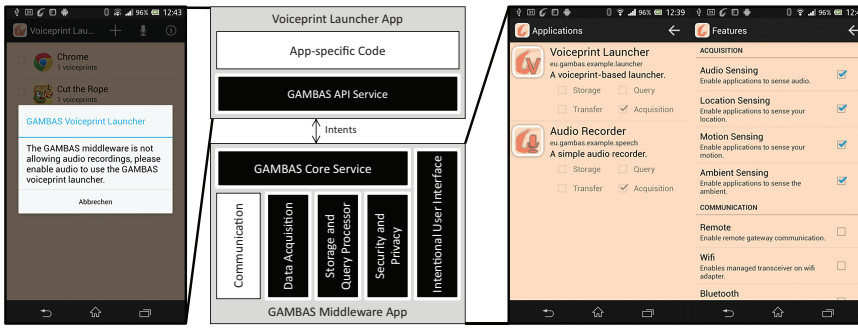


Figure 6.8 Voiceprint Launcher Coverage.

6.1.4.1.1 *Data Acquisition*

In order to capture audio data and to compute and classify voiceprints, the application uses the audio components of the context recognition framework. In particular, the application uses an AudioSensor component to capture audio, a Windowing and FFT component to perform preprocessing, a trigger component for silence detection, a voiceprint generator and matcher component for voiceprint computations and classification as well as an intent broadcaster component to signal the successful acquisition of a voiceprint as well as to signal the classification result. Figure 6.9 depicts the configurations of the component system.

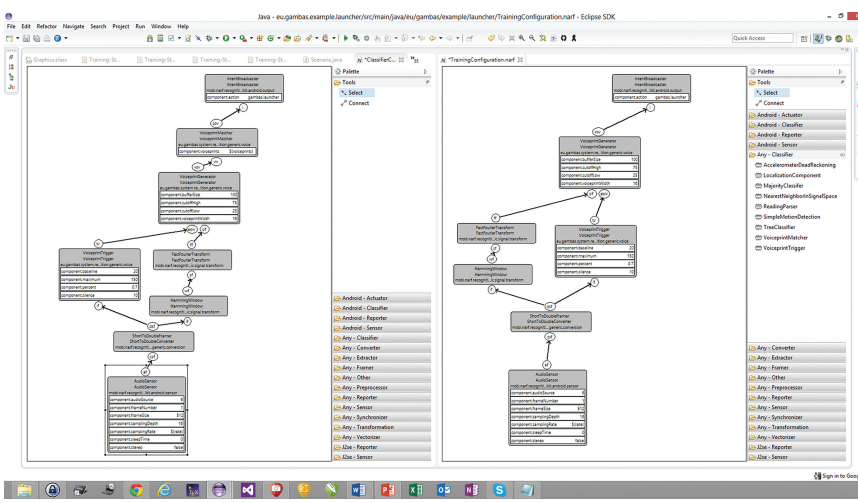


Figure 6.9 Voiceprint Launcher Configurations for Training (left) and Classification (right).

6.1.4.1.2 *Data Processing*

To store the voiceprints as well as the set of configured applications, the GAMBAS Voiceprint Launcher uses the semantic data storage as well as the SPARQL-based query processor. To store the voiceprints, they are serialized as strings such that they can be stored as RDF triples. To retrieve the set of configured applications and the associated serialized voiceprints, the GAMBAS Voiceprint Launcher uses the SDK to issue SPARQL queries against the data storage that are executed with the middleware's built-in local one-time query processor.

6.1.4.1.3 *Privacy Preservation*

Although the Voiceprint Launcher is executed locally on the device, it still integrates with some of the privacy features of the GAMBAS middleware. In particular, as depicted above, the GAMBAS Voiceprint Launcher's access to the device's soundcard and audio capabilities are controlled through the GAMBAS middleware. Thus, a user can prevent the application from recording audio by simply deactivating the associated middleware feature. Consequently, the requests to capture audio by means of the configurations depicted previously will be blocked by the middleware. The associated blocking will then be signaled back to the application via the Android SDK such that it can react to it in an adequate way, for example, by showing a dialog that tells the user that the application requires audio capabilities to function properly. Intuitively, for more complex applications, it may also be possible to provide different modes of operation, e.g., a mode that uses audio and a mode that does not. However, for the GAMBAS Voiceprint Launcher, the ability to record audio is essential. Consequently, it is not feasible to provide such a mode.

6.1.4.1.4 *Intentional User Interface*

Similar to security and privacy, the GAMBAS Voiceprint Launcher also integrates with the intentional user interface through the SDK. In order to provide the user with a clean view on the applications that are installed as well as the features that are requested by them, the GAMBAS Middleware app uses intent-based interaction to populate the list of installed GAMBAS-enabled applications. This enables the user to quickly list all GAMBAS applications and to start an application from the GAMBAS Middleware App's user interface.

6.1.4.2 GAMBAS Linked Weather

While the GAMBAS Voiceprint Launcher is focused on the Android SDK, the GAMBAS Linked Weather application focuses primarily on data management, remote communication and the J2SE SDK. The application uses the legacy data wrapper to integrate with a third-party data source, namely the weather web service provided by Wetter.com. To do this, a J2SE-based service periodically retrieves the weather information for the largest German cities and stores it in a semantic data storage that is equipped with remote communication and distributed query processing functionality such that the data becomes accessible to other devices.

To demonstrate the J2SE application as well as the interaction between J2SE-based and Android devices, we have developed a Linked Weather app. The functionality provided by the application is depicted in Figure 6.10. The application enables a user to add an arbitrary number of cities to his device. Once the cities are added, the user can press a sync button to retrieve the latest weather information. Internally, tapping the sync button will issue a series of remote SPARQL queries against the RDF data stored in the semantic data storage on the J2SE device, which will synchronize the local data storage of the Android device with the remote data storage of the server. In order to reduce the amount of data that must be transferred, however, only the cities selected by the user are actually synchronized. When the synchronization is completed, the user can tap any city to view the current forecasts. This will issue a series of local queries against the storage, to retrieve the forecasts for a city. At this point, there is no more need for remote interaction as the device already has the associated data.

As depicted in Figure 6.11, the GAMBAS Linked Weather application uses the GAMBAS middleware on Android through the Android SDK and on J2SE through the J2SE SDK. From the functionality provided by the core services, the GAMBAS Linked Weather uses four building blocks as follows:

6.1.4.2.1 *Secure Communication*

In order to interact with each other, both the Android and the J2SE parts of the application use the communication services provided by the middleware. Thereby, the Android part of the application contacts an application-specific service provided by the J2SE part of the application. To determine the communication endpoint that provides the application-specific service, the Android app interacts with the dynamic data discovery registry.

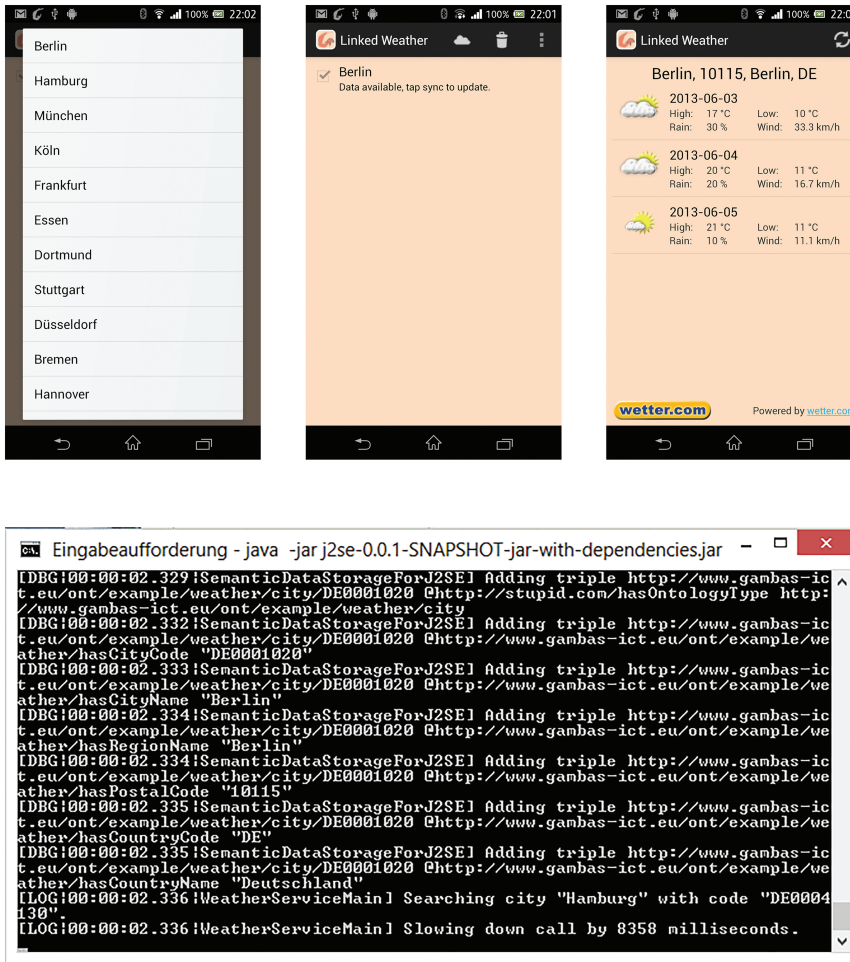


Figure 6.10 Linked Weather Android App and J2SE Service.

6.1.4.2.2 *Data Processing*

In order to store the weather information, both the Android and the J2SE parts of the application use the storage facilities provided by the local semantic data storage. In addition, the Android part of the application executes (remote) queries on the semantic data storage of the J2SE part of the application in order to synchronize the local weather information with the most current version of the weather information provided by the J2SE application.

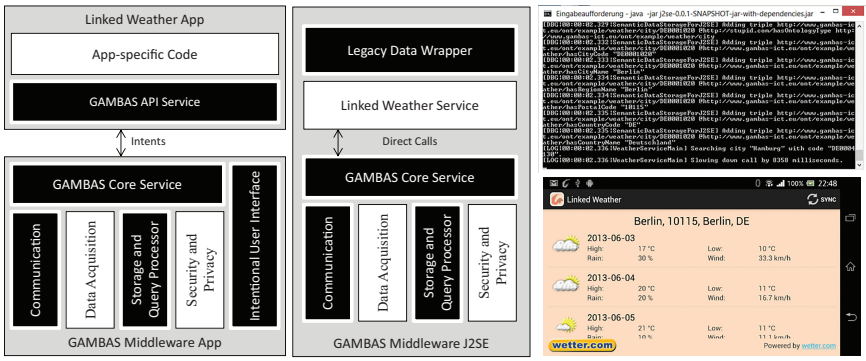


Figure 6.11 Linked Weather Coverage.

6.1.4.2.3 *Intentional User Interface*

Similar to the GAMBAS Voiceprint Launcher, the GAMBAS Linked Weather application also integrates with the intentional user interface through the SDK. The integration closely follows the explanation given previously in the sense that the application is shown in the associated list with the associated permissions.

6.1.4.2.4 *Legacy Data Wrapper*

In order to integrate with Wetter.com, the actual provider of the weather information made accessible through the J2SE service, the J2SE specific part of the application uses a legacy data wrapper that translates the custom data model used by Wetter.com to linked open data that is then stored in the semantic data storage and made available through the query processor to mobile devices. To gather data, the J2SE service periodically pull the latest data from the web service. However, in order to avoid exceeding the free quota provided by Wetter.com, the pull frequency is set to one day.

6.1.4.3 GAMBAS Locator

To demonstrate the location prediction algorithms developed as part of the data acquisition framework as well as the privacy-preserving data-sharing among devices, we have developed the GAMBAS Locator application depicted in Figure 6.12. Similar to the GAMBAS Voiceprint Launcher, the GAMBAS Locator only uses the Android version of the GAMBAS middleware. From an end-user perspective, GAMBAS Locator enables users to continuously track their location. They can track visits to locations that are relevant for them and they can share their current location with their friends

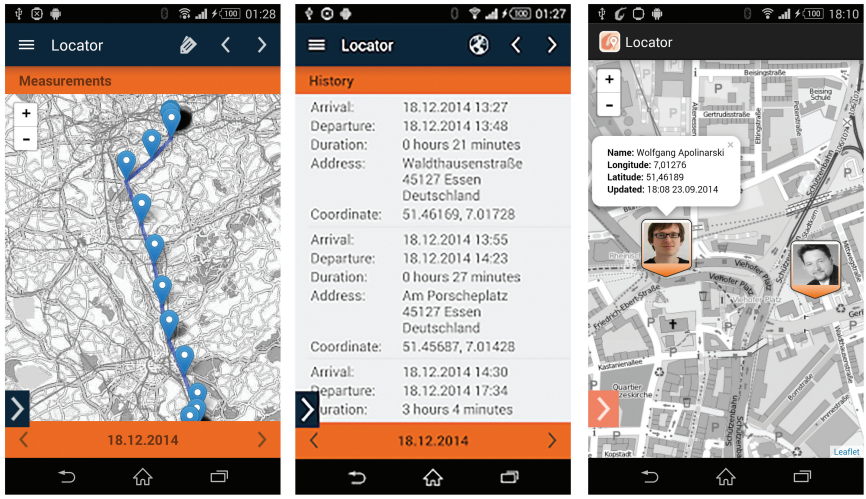


Figure 6.12 Locator History and Sharing.

in a peer-to-peer fashion through the GAMBAS middleware. In addition, the application computes and visualizes predictions for the next user location based on the location history captured by the application. Since the history and predictions are stored in the local SDS of the user's device, they can be used by other applications easily, i.e. by simply querying the local SDS.

To enable the sharing of location information with other users, the GAMBAS Locator uses the secure communication and data sharing mechanisms described in Chapter 5. To perform the necessary key-exchange for user authentication, the GAMBAS Locator can leverage the keys provided by the GAMBAS Middleware App. This means that if a user is using some social network like Facebook, for example, the user simply needs to connect the GAMBAS Middleware App with his Facebook account. Once this is done, the GAMBAS middleware will automatically exchange keys with all of his friends who are also using GAMBAS. If the user does not use social networking sites, he can alternatively use NFC to manually exchange a key. From an application programmer's perspective, using this functionality does not require a single line of code, since the middleware takes care of implementing it. Similarly, in order to share the location information with another user, the GAMBAS Locator does not require any backend service. Instead, due to the distributed processing capabilities of the middleware, the devices can exchange this information directly without a trusted third party. From an application developer's perspective, this eliminates the need and cost

for developing and running a service infrastructure. Thus, using GAMBAS, the developer can focus soley on implementing the user-facing functions.

6.1.4.3.1 *Secure Communication*

In order to interact with each other, the Android applications of different users are relying on the secure communication services provided by the middleware. Thereby, the authentication and encryption is done transparently for the application. In addition, it is noteworthy to point out that the sharing is not mediated through a service, which is the common realization of most location sharing apps that are available today. The keys that are required to ensure a proper end-to-end authentication of different users are provided by the mechanisms of the privacy preservation framework.

6.1.4.3.2 *Data Acquisition*

To capture the user's location, the GAMBAS Locator application makes use of the data acquisition framework. To do this, it sets up an component configuration with an associated state machine in the activation system. Together, the component and the activation system perform a periodic but energy-efficient localization of the user's device. To do this, the localization stack integrates the motion sensors, the GPS receiver and the network hardware. This ensures that the energy-hungry GPS receiver is only used when the user's location cannot be established through the motion sensors or through Wi-Fi scans. In addition, the GAMAS Locator also uses the data acquisition framework to perform the predictions on the user's next location. The predictions are triggered periodically whenever a new location is detected. Towards this end, the prediction components are accessing the user's location history that is stored in the semantic data storage of the user's device.

6.1.4.3.3 *Data Processing*

In order to store the location information including the user's location history and the predicted next location, the GAMBAS Locator leverages the Semantic Data Storage of the device. When a prediction must be computed, the prediction components query the local data storage for an (aggregated) view on the user's history. The data model presented in Chapter 4 specifically addresses this issue by supporting aggregation.

6.1.4.3.4 *Privacy Preservation*

To enable access control on the data stored in the SDS, the issuer of remote queries must be authenticated. To do this, the privacy preservation framework

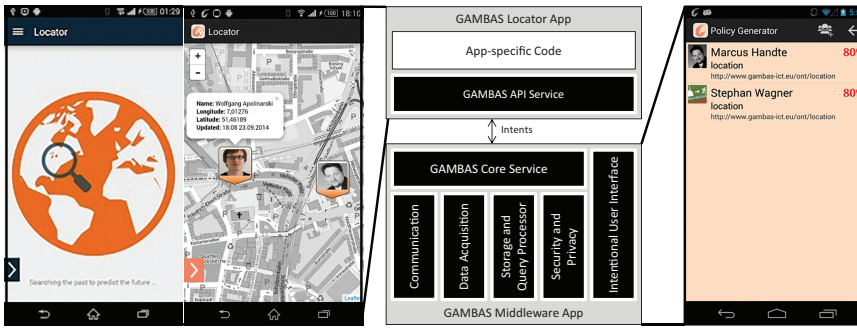


Figure 6.13 Locator Coverage.

presented in Chapter 5 defines two key exchange mechanisms that are either automatic (i.e. when using Piggybacked Key Exchange on top of an online service) or easy to use (e.g. when using a physical gesture to exchange a key between two nearby devices through NFC). In addition to authenticity, however, it is also necessary to define who should be able to access the data. For this, the privacy preservation framework provides an automatic policy generation tool that provides a pre-configured privacy policy based on the user's sharing behavior. Based on this, the user can get recommendations (c.f. Figure 6.13) for suitable policies that can be customized later on. Due to these two mechanisms, the GAMBAS Locator application does not need to handle the intrinsics of secure sharing. Instead, it simply relies on the GAMBAS middleware which automatically provides the necessary mechanisms to enforce the level of privacy desired by the user.

6.1.4.3.5 *Intentional User Interface*

Similar to the other applications, the GAMBAS Locator also integrates with the intentional user interface through the SDK. The integration closely follows the previous explanations. However, due to this integration, the application developer does not need to provide user interfaces to configure the sharing of location information. Instead, the application can simply rely on the definitions managed through the user interface of the GAMBAS Middleware app.

6.2 Application Architecture

As basis for the description of the application components in the next section, we provide an instantiation of the high-level architecture detailed in Chapter 2

for the mobility and environmental scenario outlined in Chapter 1. For each of the scenarios, we describe deployment that maps the abstract software components detailed in the component view to concrete systems. Furthermore, we outline the interactions that will take place at runtime.

6.2.1 Mobility Scenario

As described in Chapter 1, one of the motivating application scenarios behind the GAMBAS middleware is support for mobility applications in a smart city. To demonstrate the middleware capabilities, we developed a so-called Public Transport Exploitation System (PTES) and a GAMBAS mobile application to take into account the information retrieved directly from the user and to offer citizens customized services – not exclusively related to mobility though – in order to enhance their trip experience. Overall, the scenario encompasses personal mobile Internet-connected objects such as the smart phones of citizens, buses that are equipped with embedded systems, existing external services and a number of novel GAMBAS services. Figure 6.14 shows both their deployment and interaction.

6.2.1.1 System Deployment

As depicted in Figure 6.14, the mobility scenario contains a number of computer systems that run various parts of the GAMBAS middleware as well

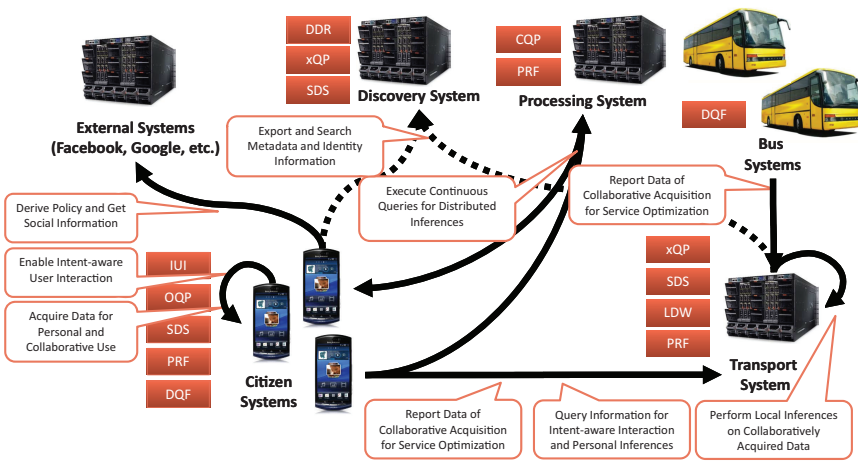


Figure 6.14 Mobility Scenario Architecture.

as application-specific code that realizes the selected use cases. The computer systems are:

- **Citizen Systems:** In order to access services, citizens make use of their personal mobile devices like smartphones and tablets, which are running a mobile application. The application consists of the intent-aware user interface as well as background services that automatically acquire data and either forward or store it. Furthermore, it makes parts of the stored data accessible to other devices. Typically, these systems can be considered as Constrained Computer Systems (CCS). Consequently, the background operations must optimize their resource usage, especially in terms of energy. The application makes use of the data acquisition framework, the semantic data storage and – to protect the user's privacy – the privacy preservation framework. In order to make data available to other devices and to support local inferences at the application level, the device is equipped with a one-time query processor. Finally, in order to enable intent-aware user interaction, the application makes use of the intent-aware user interfaces.
- **Transport System:** In order to provide route information and to aggregate capacity-related information, we introduce a transport system that is available on the Internet. Since the data about bus routes and schedules is already available in a legacy system, the transport system uses a legacy data wrapper in order to tap into this information source. Furthermore, in order to store information coming from the citizen systems as well as from public buses, the system is equipped with semantic data storage. To support local as well as distributed inference on the data and to make it available to third parties, the system is equipped with a one-time and a continuous query processor. Finally, in order to protect the raw data and to restrict the sharing of data, the system is equipped with a privacy framework component that limits the sharing accordingly.
- **Bus Systems:** Besides the citizen systems, the mobility scenario also relies on embedded systems deployed in public buses in order to collect data. Consequently, the buses are equipped with an application that determines the relevant context and forwards it to the transport system, which then stores and aggregates the data. In order to do this, the embedded system running in the bus makes use of the data acquisition framework in order to acquire and report the data.
- **Discovery System:** To enable transparent access to data coming from different data sources, it is necessary to make the possible data sources

discoverable. Performing this task is the primary function of the discovery system. In order to do that, it runs a data discovery registry which uses the semantic data storage component and a one-time as well as a continuous query processor component in order to store metadata and identity information of data sources. In contrast to other systems in the architecture, this system is application-independent.

- **Processing System:** To enable the citizen systems to run continuous queries against each other's devices, the architecture encompasses a second generic type of system. This processing system is equipped with a privacy framework and a continuous query processor.
- **External Systems:** To reduce the configuration effort for the privacy mechanisms, the privacy preservation framework taps into the information available in other external systems. For this, the privacy framework provides a number of adapters that can access the user-specific information in these external systems. Since these systems are maintained by third parties, no additional GAMBAS software is installed on them. Consequently, the adapters of the privacy framework are responsible for performing the necessary data conversion.

6.2.1.2 System Interaction

In order to implement the mobility scenario, the systems and their associated components have to interact with each other locally (within a single system) and some of them have to interact remotely. This interaction follows the abstract interaction patterns described as part of the dynamic perspective in the high-level architecture presented in Chapter 2.

In order to enable distributed query processing, all semantic data storage components export metadata and/or identity information to the discovery system. The query processors and the privacy framework use this information transparently to determine and contact the appropriate data sources and to create the necessary views, respectively.

For the mobility scenario, most queries are issued by the citizen systems. They target either the transport system, e.g. in order to compute route information, or other citizen systems, e.g. in order to find collocated routes or to determine whether two friends are in the same bus. Since some of the latter type of queries may be continuous queries, the remote processing system must interpret them – as continuous queries are not supported directly on Constrained Computer Systems (CCS).

In order to provide advanced behavior-driven services, the citizen systems and the bus systems are used to collect data collaboratively. As described

previously, for citizen systems, this requires each citizen to opt in to the data collection and sharing by configuring the appropriate privacy settings. For bus systems, such a configuration is not necessary since the collected data does not affect privacy. Once relevant data is collected at the bus system or the citizen system, it is reported to the transport system.

The transport system collects the data received from the bus systems and citizen systems. Furthermore, it stores and aggregates it for service optimization purposes. This should typically result in local inferences as the aggregations required for the mobility scenario do not require dynamic data that is not available locally.

In order to make the optimized services accessible to the citizens, the application running on citizen systems provides an intent-aware user interface. Using the behavior information gathered by the data acquisition framework, the intent-aware user interface can notify the citizen about important events and it can display relevant information at the right time. In cases where the required predictions for this are imprecise or not possible, the citizen may specify goals using a speech recognition engine that is part of the framework.

In order to fetch the information that is relevant for the citizen, the intent-aware user interface issues queries and performs local or distributed inferences using the query processor and application-specific code. For some distributed inferences, it is necessary to access the data gathered by citizen systems of other citizens that share this data.

In order to enable privacy-preserving sharing, the privacy preservation framework controls the access to the data stored on the citizen systems. The basis for this is a privacy policy that is initialized using the information from external services such as Facebook or Google. The privacy preservation framework retrieves the privacy-related information from these systems periodically in order to determine relationships between different citizens and to keep the initial policy up-to-date. However, it is noteworthy that citizens can manipulate this generated policy through the user interface in order to customize it to their needs.

6.2.2 Environmental Scenario

The environmental application scenario is related to the mobility scenario due to the sources of information that are used for data collection. Specifically, the architectural instantiation described in the following relies on the data being captured by the bus system and a mobile application. Consequently,

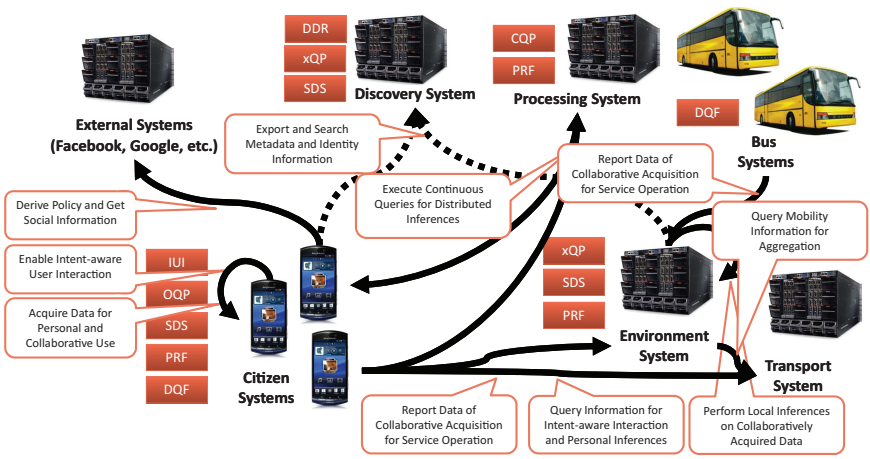


Figure 6.15 Environmental Scenario Architecture.

from an architectural perspective, the environmental scenario can be thought of as an extended version of the transport scenario. This is also clearly visible when comparing the instantiated architecture depicted in Figure 6.15 with the associated instantiation of the mobility scenario depicted in Figure 6.14. Nonetheless, we briefly describe both the deployment and the resulting interaction. For the sake of brevity, we refrain from revisiting the interactions with the transport system and focus on the environment system instead.

6.2.2.1 System Deployment

As depicted in Figure 6.15, the environmental scenario contains a number of computer systems that run various parts of the GAMBAS middleware as well as application-specific code that realizes the application functions. As indicated previously, a number of these systems are slight variations of the systems in the mobility scenario:

- **Citizen Systems:** Citizens are using their systems to gather information and to access services. For this, they rely on the same set of components as in the mobility scenario. However, the application-specific code has to be extended to accommodate the different usage.
- **Transport System:** Since some of the environmental use cases require transport-related information, the transport system of the mobility scenario is used to offer it. Specifically, the transport system is used to determine bus locations, which are required to provide the necessary context for environmental information.

- **Bus Systems:** Similar to the mobility scenario, the environmental scenario also makes use of embedded systems deployed in public buses in order to collect environmental data. The environmental data, however, will not be reported to the transport system but it will be reported to a new system – called the environmental system.
- **Discovery System:** Since the environmental scenario also requires distributed data processing, it is necessary to rely on the discovery system that manages the metadata and identity information.
- **Processing System:** Just like in the mobility scenario, a dedicated processing system is used to enable the citizen systems to run continuous queries against each other's devices. The processing system is equipped with a privacy framework and a continuous query processor.
- **External Systems:** The environmental scenario makes use of external systems to initialize the privacy policy. These systems are maintained by third parties, so no additional GAMBAS software can be installed on them and the necessary adapters are provided by the GAMBAS middleware app.

In addition to these systems which were also used for the mobility scenario, the architecture of the environmental scenario also introduces a new system:

- **Environment System:** Conceptually, the environment system is related to the transport system introduced in the mobility scenario as it manages and aggregates the environmental data reported by the bus and citizen systems. The main difference between the transport system and the environment system is the lack of a legacy data wrapper since the environment system does not have to tap into existing data sources. Other than that it, performs conceptually similar tasks such as data storage, aggregation and the computation of inferences.

6.2.2.2 System Interaction

In order to realize the different applications for the environmental scenario, the systems and their associated components have to interact with each other in a similar fashion as in the mobility scenario. The export of metadata and identity information is handled by the data storage components, the privacy framework and the data discovery registry. The searching is done transparently by the query processors and the privacy framework, which also create views on the semantic data storage of the citizen system, if necessary. The control for this is enabled by the policy generated by the privacy framework, which can be manually adjusted through the user interface.

Queries are issued by the citizen systems and the environmental system. They target either the transport system, e.g. in order to compute route information or other citizen systems. If a citizen system requires the execution of a continuous query, the remote processing system is used.

In order to provide advanced behavior-driven services, the citizen systems and the bus systems are used for collecting data collaboratively. The environment system collects the data received from the bus systems and citizen systems. Furthermore, it stores and aggregates the data, for example, to offer a pollution map, which can be used in conjunction with the transport system to compute alternative routes. This should typically result in local inferences at the environment system since the route information is mostly static and can be retrieved once for each computation.

In order to make the environmental services accessible to the citizens, the application running on citizen system provides an intent-aware user interface. In order to fetch the information that is relevant for the citizen, the intent-aware user interface issues queries and performs local or distributed inferences using the query processor and the application-specific code. This may entail distributed inferences, which are enabled by combining the continuous query processor on the processing system with the privacy frameworks on the citizen systems.

6.3 Application Components

As indicated by the application architecture, the implementation of the application scenarios entails a number of different components that are required to deliver the application functions. In the background, there are a number of application services that store and offer the data captured through sensing applications used by citizens or running in buses. In addition, there are background services that wrap legacy data coming from third-party data sources. Thus, in order to create a complete picture of the applications developed as part of GAMBAS, we first describe these application services. Thereafter, we outline the applications that we developed to capture the required data. On the basis of this description, we then describe the end-user applications for citizens as well as a set of innovative applications that feed the captured data back to the transit network operator.

6.3.1 Application Services

To power the mobility and environmental applications, we have developed a number of application-specific services using the GAMBAS middleware.

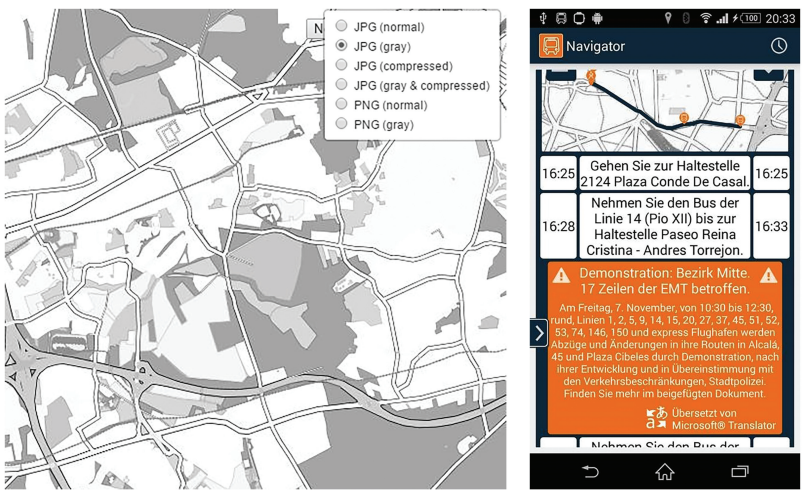
These services integrate with different data sources including data coming from EMT Madrid (incident feed, time tables, routes, etc.), open data provided by OpenStreetMap (addresses, geometry, etc.) and application-specific data (e.g. crowd-levels measured by the embedded applications running in vehicles). Although these services are conceptually backend services that are not directly visible to end users, the application services encompass frontends targeted at application developers and service administrators. In the following, we briefly walk through the different services and, where applicable, show a few screenshots of their frontends.

6.3.1.1 Tile Service

The tile service integrates with OpenStreetMap geometry data in order to generate images that are used to draw the map-based visualization. It supports multiple output formats and color schemes. The api has been designed to work with the Leaflet.js Javascript library, which is used consistently throughout the GAMBAS mobile applications. The screenshot shown in Figure 6.16(a) depicts a number of output options.

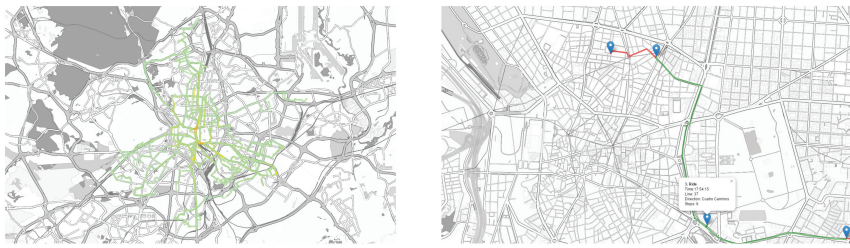
6.3.1.2 Incident Service

The incident service integrates with the EMT Madrid incident feed in order to provide incident information to the navigation application described later on.



(a) (b)

Figure 6.16 Tile and Incident Service. (a) Tile Service and (b) Incident Service.



(a) (b)

Figure 6.17 Crowd and Routing Service. (a) Crowd Service and (b) Routing Service.

It is tightly integrated with the routing service in order to enable the output of route incidents for trips computed by the user. Since the EMT incident feed is only available in the Spanish language, the incident service has been integrated with Microsoft Translator, which provides machine translations into other languages supported by the mobile prototype applications. Figure 6.16(b) shows the resulting machine-translated output that is integrated into the routing result on the mobile app.

6.3.1.3 Crowd Service

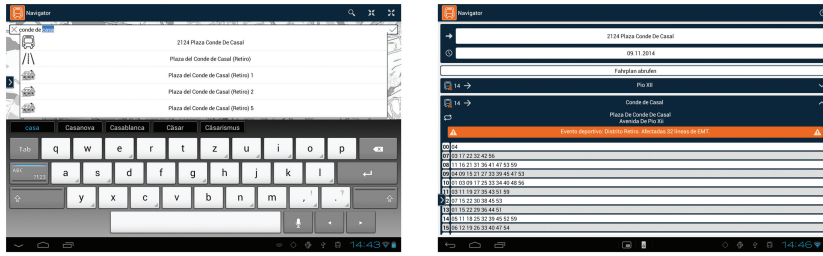
The crowd service captures the crowd-level information generated by several buses in the city of Madrid. The captured data is then used by the routing service in order to provide crowd-level information as part of the routing result. To do this, the service aggregates the reports and assigns them to 15 minute timeslots, which are then used to drive the predictions. Figure 6.17(a) shows a sample crowd-level for one of these 15 minute timeslots.

6.3.1.4 Routing Service

The routing service (c.f. Figure 6.17(b)) integrates with the EMT GTFS data in order to compute crowd-aware routes, which are then used to power the navigation functions in the mobile application for citizens. In addition to transit routes using buses, it can also compute walking routes. If available, incident and crowd-level data will be returned directly as part of the routing result in order to minimize the amount of data that must be transferred between the mobile application and the service.

6.3.1.5 Network Service

The network service provides the mobile application for citizens with network-related information such as location and names of stops, routes of



(a) (b)

Figure 6.18 Network and Timetable Service. (a) Network Service and (b) Timetable Service.

lines, etc. The resulting information is then used to visualize the route in the application. Using a tool, it is possible to extract the information from the network service and to ship it with the application. This is done in order to minimize the latency for displaying search results. Figure 6.18(a) shows the application which uses the built-in address database for auto-completion during place search.

6.3.1.6 Timetable Service

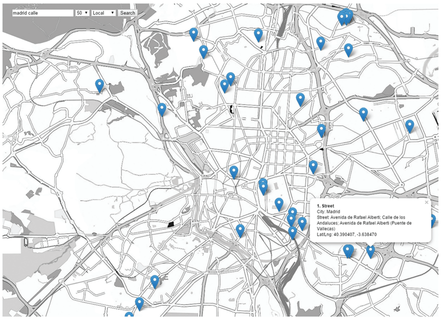
The timetable service provides the mobile navigation application with bus schedule information that is extracted from the GTFS information provided by EMT Madrid. Since the associated amount of GTFS data is too large to be processed directly on the device, this service takes care of extracting the relevant subsets based on a stop name and a calendar date. The mobile application then visualizes the output in a tabular form as depicted in Figure 6.18(b).

6.3.1.7 Geo Service

The geo service integrates with OpenStreetMap in order to resolve addresses into GPS coordinates. The service is used by the mobile applications to enable the user to search for addresses and to resolve GPS coordinates into addresses. The number of results returned by the service to the application is configurable in order to enable the optimization of applications for different criteria (i.e. bandwidth vs. flexibility). Figure 6.19(a) shows the output of the service on a map when searching for a particular address.

6.3.1.8 Log Service

The log service captures usage information generated by the mobile applications and enables the offline analysis of the user behavior for evaluation



(a)

Component	Uid	First	Last	Count
eu.gambas.prototype.locator	36c1c4a2-2626-4842-9fb2-c3119901ba63	30.12.14 23:33:47	31.12.14 15:34:21	1740
eu.gambas.prototype.locator	4840d1aa-2fb6-44f9-993b-21eeb816860d	31.12.14 16:54:30	02.01.15 16:34:22	4610
eu.gambas.prototype.navigator	0024de45-e059-4467-90bd-bfdd02bc2e39	21.11.14 18:25:45	21.11.14 18:25:45	1
eu.gambas.prototype.navigator	0a00004f-a49e-4a60-b061-9a6a57cf0123	26.11.14 09:13:57	11.12.14 14:20:05	18
eu.gambas.prototype.navigator	0dd77ddb-657d-4c32-8b57-ebb09b44e815	05.11.14 08:27:19	14.11.14 13:40:31	170
eu.gambas.prototype.navigator	0ecb4f5a-6b1d-4b18-825e-04bc16548a4b	26.11.14 19:40:36	26.11.14 20:16:00	10
eu.gambas.prototype.navigator	1367fa7b-e2c4-4e08-8070-0162a663f6b4	11.12.14 14:29:55	11.12.14 18:10:50	28
eu.gambas.prototype.navigator	181e9e11-6c98-447d-8f6f-ddf7fff6117e	25.11.14 19:02:57	26.11.14 17:28:27	50
eu.gambas.prototype.navigator	3707aaf0-9eaa-4ec2-b769-cb747ff7bf8f	02.11.14 21:01:15	02.11.14 21:01:15	1
eu.gambas.prototype.navigator	3f3a4d99-25b2-4511-818c-d65034b10d04	07.11.14 00:43:52	16.11.14 14:08:28	214
eu.gambas.prototype.navigator	3fad1b68-f6f9-4158-a603-f37e66c0b1d5	18.11.14 15:50:48	25.11.14 16:52:12	1356
eu.gambas.prototype.navigator	4971e348-7f48-43db-bd8b-a1d25477f87f	06.11.14 23:32:09	06.11.14 23:32:09	1
eu.gambas.prototype.navigator	4c245ae1-8277-4a3a-b397-0506eaed4d6e	17.11.14 21:48:27	18.11.14 04:20:48	4
eu.gambas.prototype.navigator	4e396854-c89c-492e-843a-ea10e2de074b	21.11.14 11:25:11	15.12.14 13:52:13	15
eu.gambas.prototype.navigator	532a99c6-045f-46ad-9504-4f5be5a1972b	05.11.14 18:15:12	24.11.14 23:07:53	35
eu.gambas.prototype.navigator	53863bf1-02af-4f60-a271-6aab871498cf	20.11.14 16:34:39	21.11.14 14:21:57	48
eu.gambas.prototype.navigator	560e0a4c-0e1f-428d-b392-43dc9b7c6556	26.11.14 22:09:34	02.12.14 15:26:39	38
eu.gambas.prototype.navigator	579ff7f3-3d1b-432f-ad6e-72d61a59410c	17.11.14 16:33:25	29.11.14 00:40:45	3
eu.gambas.prototype.navigator	5fde1318-77fb-46d4-b2a9-1cae888ef329	07.11.14 22:47:38	07.11.14 22:47:38	1
eu.gambas.prototype.navigator	619e5a32-73bd-4cb2-b1f7-c942ce09b02a	17.12.14 14:49:32	17.12.14 14:50:50	6
eu.gambas.prototype.navigator	6741e19e-868a-45c4-9fd7-e8e8d0c7d804	26.11.14 21:23:08	27.11.14 11:06:10	10
eu.gambas.prototype.navigator	7a6b3de7-bd43-404a-8998-f5ab5e40f895	25.10.14 12:50:35	29.10.14 11:09:32	2559
eu.gambas.prototype.navigator	846c2b55-b0c1-4849-bd98-8c61833e1e61	26.11.14 18:23:45	26.11.14 18:23:45	1
eu.gambas.prototype.navigator	95dd7f25-db1e-4b28-a866-1921fd0a6b1c	23.11.14 23:04:12	23.11.14 23:04:12	1
eu.gambas.prototype.navigator	97814df0-3bc0-457c-b149-4a52db5bb82a	20.12.14 13:45:05	20.12.14 13:45:05	1
eu.gambas.prototype.navigator	9b3b0ab5-9aa0-4ffc-baa3-91aaf4542f8c	29.10.14 11:52:38	29.10.14 15:44:36	4
eu.gambas.prototype.navigator	a6269923-316c-46d0-a473-63c90967659b	03.11.14 06:43:16	03.11.14 06:43:16	1
eu.gambas.prototype.navigator	a6d078ef-1cdf-4084-9140-5db7bc5f0eba	29.11.14 10:40:30	29.11.14 10:40:30	1
eu.gambas.prototype.navigator	ad780f5c-5f16-4d87-9088-a66c24847f64	29.10.14 16:18:45	29.10.14 16:20:57	2
eu.gambas.prototype.navigator	b649a705-d579-4977-91e5-b5925b495d7c	28.10.14 16:16:14	28.10.14 16:56:14	4
eu.gambas.prototype.navigator	b7a9dfcc-d67a-4d3d-8a95-393fe7544ea8	01.12.14 17:43:46	01.12.14 17:43:46	1
eu.gambas.prototype.navigator	b9b790a8-d110-4a07-a9b6-001a47ff5add	20.11.14 15:08:46	24.11.14 17:48:13	2
eu.gambas.prototype.navigator	c9dbcb7d-47db-4b5d-b498-c64cd4d64c12	13.11.14 09:29:26	18.12.14 17:17:16	431
eu.gambas.prototype.navigator	d2cfe6f4-3c8d-48c0-8c73-2d4a5d213558	01.01.15 23:35:54	01.01.15 23:35:54	1
eu.gambas.prototype.navigator	d846a5dd-1534-4072-8184-7d064f7ed2d3	30.11.14 01:05:00	30.11.14 01:05:00	1
eu.gambas.prototype.navigator	e254c9f9-c5f1-4eee-bd60-e38854700485	03.11.14 09:43:02	30.12.14 06:56:02	35
eu.gambas.prototype.navigator	ecea83ea-f58f-448c-b6e4-9ab3cf5d88af	18.11.14 11:00:31	18.11.14 19:13:34	2
eu.gambas.prototype.navigator	eddd53d7-1ded-46ca-8184-457b0e6d7ccd	25.11.14 17:13:36	25.11.14 17:13:36	1
eu.gambas.prototype.navigator	f06603ab-aed8-4447-808d-80ef08bc7933	29.10.14 18:31:08	11.11.14 14:15:18	361
eu.gambas.prototype.navigator	f180d2f1-3645-48d0-8b02-7d52e062285a	29.10.14 19:15:10	04.11.14 22:36:12	52
eu.gambas.prototype.navigator	f8fe03d9-6ba4-44ae-9f11-9aa1229852ec	15.11.14 15:16:26	15.11.14 15:16:26	1
eu.gambas.prototype.navigator	fd1469f5-ba39-4c26-a2f2-91d8149a0add	29.10.14 13:04:18	29.10.14 13:07:03	2
		25.10.14 12:50:35	02.01.15 16:34:22	11823

(b)

Figure 6.19 Geo and Log Service. (a) Geo Service and (b) Log Service.

purposes. It is based on a simple event abstraction that captures the reporting component, a generated user identifier, the time and the type of event as well as associated application-specific event data. A logging framework that has been integrated into the mobile applications is used to capture and synchronize the data produced by an application with the service. The captured events can be downloaded for later analysis. To do this, the service supports different queries based on event types, application components, dates, etc. In addition, the service can generate a report summary to track its internal status (e.g. which devices are uploading data, when devices have uploaded data and how much data has been captured already).

6.3.1.9 Noise Service

This service captures, aggregates and visualizes the noise information captured by a mobile sensing application. In addition, it can display the noise level for the captured locations. In order to avoid overloading clients, the data is aggregated inside the service before it is delivered to other applications. Figure 6.20 shows some sample noise data. The circles indicate locations where noise measurements are available. The circle color indicates the average noise-level at the location.

6.3.1.10 Environmental Service

The environmental service captures the environmental information gathered through measurements taken by sensors located in various buses that are driving through the city of Madrid. Thereby, the service associates the measurement with the real-time location of the bus. The service is equipped with

Figure 6.20 Noise Service.

Figure 6.21 Environment Service.

a simple user interface so that the individual measurments can be displayed. Thereby, it is possible to filter the measurements based on the sensor type as shown in Figure 6.21.

6.3.2 Sensing Applications

To provide data for the end-user applications, we have developed a number of sensing applications that target environmental information (noise, CO2-level, pollen-levels, etc.) and transit information (i.e. crowd-levels of buses). As indicated in Section 6.2, this data is captured partly by mobile applications running on the devices of end-users and partly by embedded applications that

are integrated into the buses that are operating in the city of Madrid. In the following, we briefly describe these sensing applications.

6.3.2.1 Noise-level Mobile App

To measure the noise profile at different points in the city, we extended the GAMBAS Locator application described in Section 6.1.4.3 with support for crowd-sensing. To do this, we integrated a data acquisition configuration that captures the sound profile using as the average frequency vector described in Chapter 3 and the sound pressure level. A user that wants to participate can activate the periodic background capturing of the sound profile through a settings screen. When activated, the sound profile is stored locally whenever the user's location is computed. As a result, the user can then visualize the the daily noise exposure as shown in Figure 6.22.

In addition to locally storing the information, the user can opt-in to crowd sensing. If the user enables crowd sensing, the sound profile and noise level will be uploaded to the noise service together with the user's current location and measurement time. Given a larger number of participants, the measurements can be used to create a picture of the noise profile of a city.

6.3.2.2 Pollution-level System

To capture pollution information in the city, we equipped a small number of buses with an environmental sensor as shown in Figure 6.23. Using an

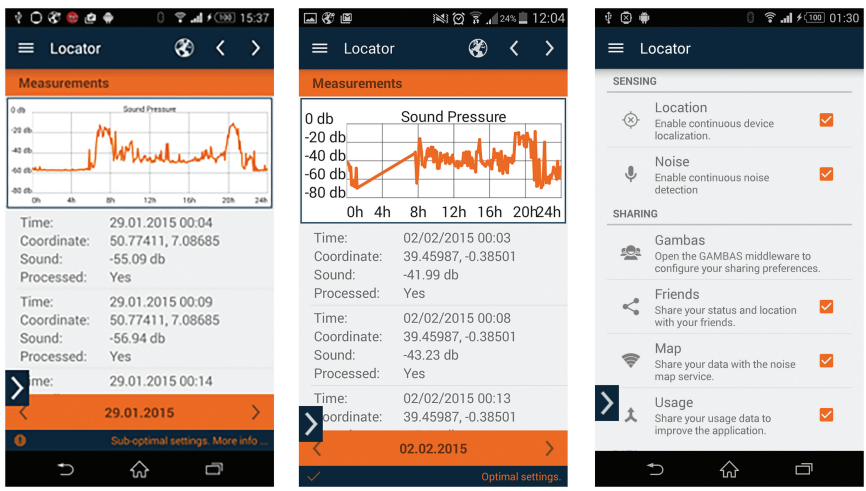


Figure 6.22 Noise-level Crowd-Sensing.


Figure 6.23 Embedded Sensors.

application embedded into the existing ICT infrastructure of the bus, we were using this deployment to continuously capture sensor readings while the bus was operating. The captured readings were then transmitted to the environmental service where the sensor data was stored together with the real-time GPS position of the bus.

6.3.2.3 Crowd-level System

One of the innovative functions of the mobile navigation app described in the following is to provide users with real-time and predicted crowd-level information about the vehicles on different routes. To capture this crowd-level information, we developed an embedded system and integrated into several buses [HIW+14]. The system consists of a TP-Link 3020, which is equipped with a Linux-based operating system (OpenWRT) running the JamVM virtual machine. Several operating system services have been specifically configured to enable a simple installation (e.g. DHCP, NTP) and to support remote administration (e.g. SSH, AutoSSH). The system uses pcaplib and tcpdump in order to sniff 802.11 probe requests and beacon frames. These are then interpreted by a set of components running on top of the GAMBAS data acquisition framework in order to determine the crowd-level of a bus (by counting the number of people). Figure 6.24 depicts the hardware as well as the software stack.

The configuration depicted in Figure 6.25 consists of a number of components. The first one (RadioTap Sensor) captures packets using tcpdump, filters and classifies them. Sitting on top of the senor, the annotator and gate components are responsible for counting the persons. Finally, the reading segmenter prepares an output file to be transmitted to a server at regular time intervals. These uploads are then performed asynchronously using the reading uploader. The reading uploader interacts with the crowd service, described previously, that stores the crowd-levels and makes them available

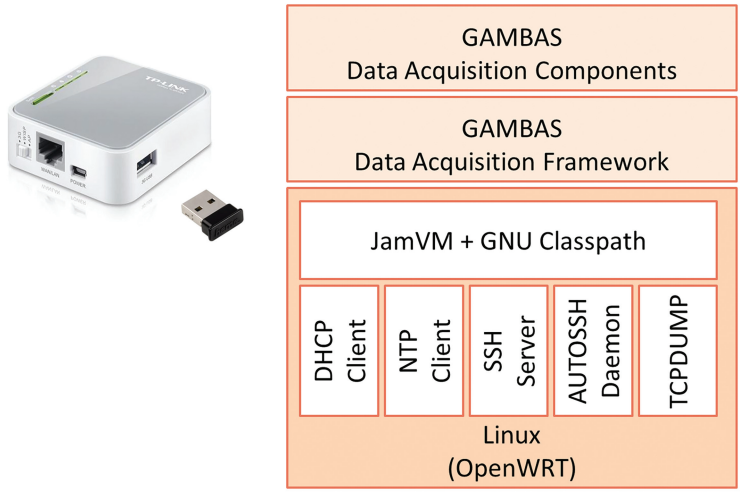


Figure 6.24 Embedded Crowd-Level Detection Application.

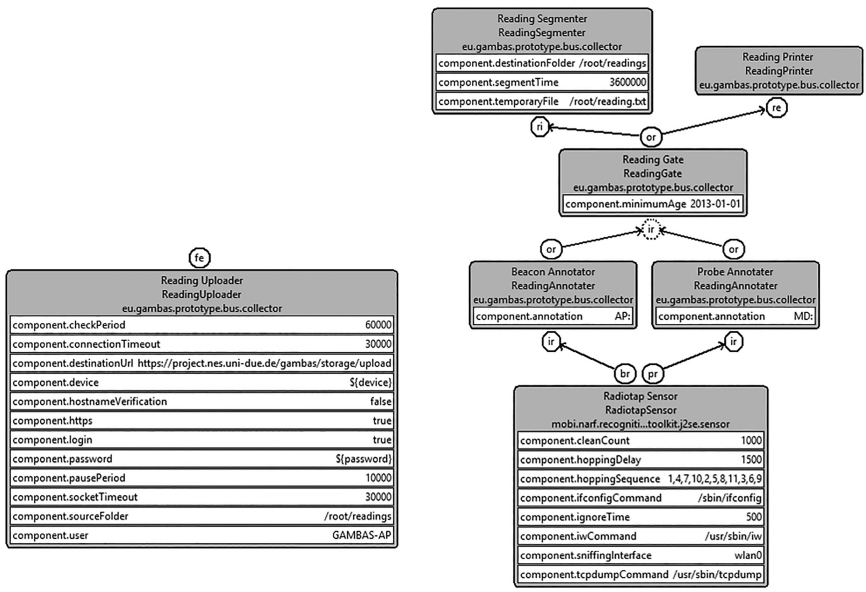


Figure 6.25 Crowd-Level Detection Configuration.

to the mobile application. In order to mitigate potential privacy issues, however, the collected data is anonymized by removing any personal identifiable information (i.e. device MAC addresses) before it is uploaded.

6.3.3 End-user Applications

The end-user applications provide regular citizens with the ability to access the information captured through the sensing applications and managed by the GAMBAS application services. In the following, we briefly outline the two end-user applications that have been developed for the mobility and the environmental scenario.

6.3.3.1 Navigation App

For the mobility scenario, we developed a mobile application for Android. Since the mobility-related application services are integrating data from the public bus network of the city of Madrid, we called this application **Madrid Navigator.**

The Madrid Navigator is a maps and navigation application that is conceptually similar to other modern navigation applications for mobile phones. As depicted in Figure 6.26, it provides users with a map of their environment and allows them to search for places and bus stops. Using the voice control

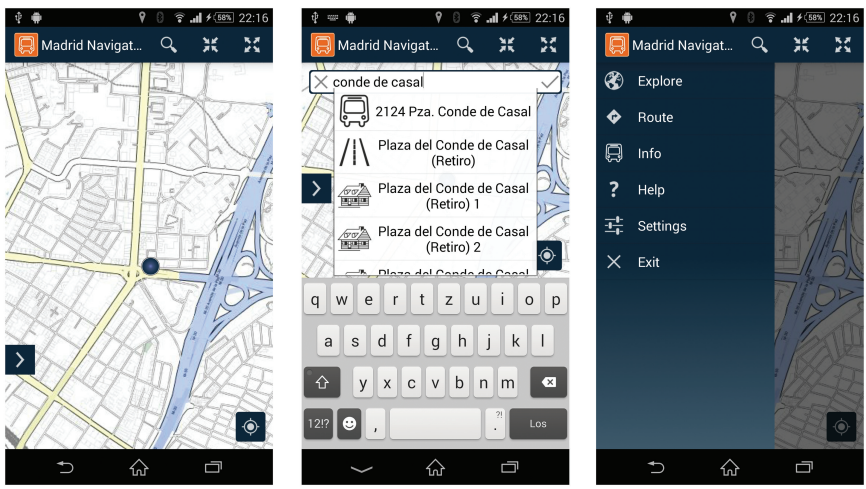


Figure 6.26 Madrid Navigator App. (a) Position, (b) Search and (c) Menu.

components described in Chapter 3, users can not only search for places via text input but also through speech input. Using different icons, the Madrid Navigator categorizes search results into cities, streets, buildings and bus stops. Depending on the category, the Madrid Navigator can show additional information such as bus routes going through a particular stop or timetables.

In addition to retrieving additional information, the search results can also be used to compute routes. To do this, a user can simply pick any place on the map and tap on a route button. Alternatively, the user can enter a source and a destination address or GPS coordinate into the routing screen shown in Figure 6.28. On this screen, a user can also adjust different parameters such as the desired arrival or departure time and specify the desired modality (e.g. on foot or by bus). When the user is satisfied and starts the computation, the specified parameters are transmitted to a GAMBAS service that computes one or more route alternatives. Once the routes have been computed, they are visualized in a list of route summaries. The user can inspect this list and get additional information by tapping on one of the summaries.

As shown in Figure 6.27, the detailed view not only shows information about the sequence of actions on the route, but it also depicts crowd-level information on the specific bus that is proposed. This allows users to compare consecutive trips on the same route with respect to the expected crowdedness

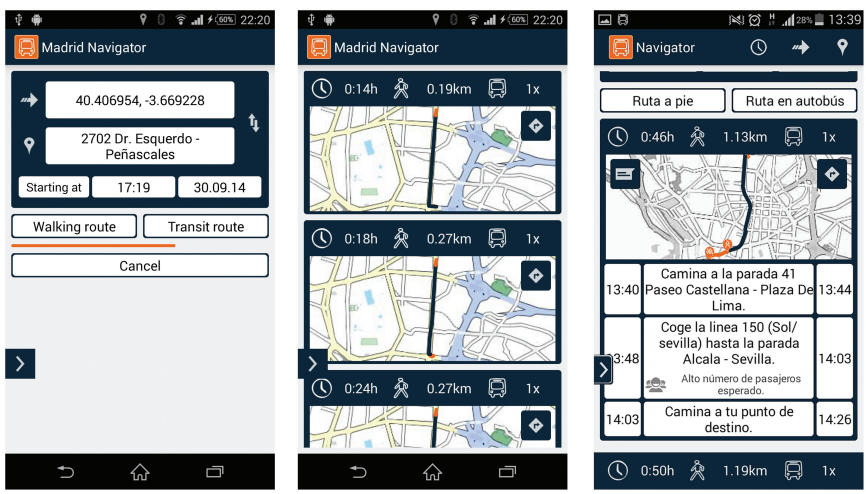
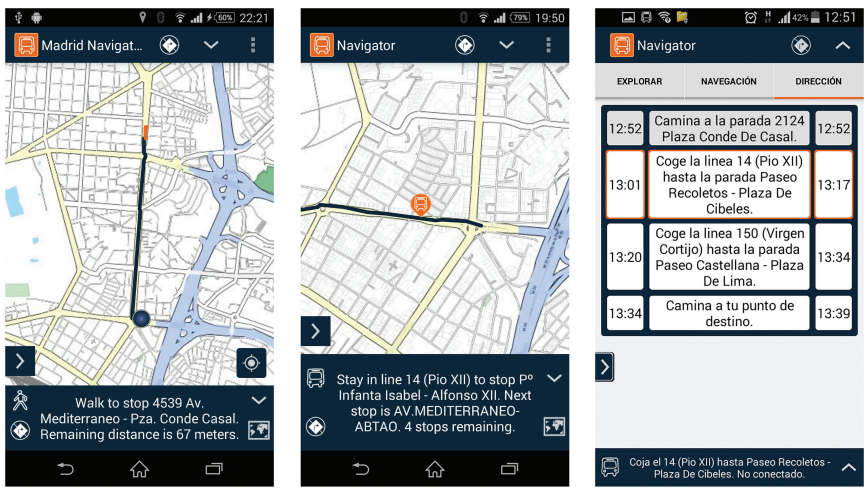


Figure 6.27 Madrid Navigator Routing. (a) Request, (b) Summary and (c) Detail.

of the bus. To determine this information, we use the embedded sensing application described previously, which we deployed in several buses running through the city. Using this embedded application, we collect real-time information about the number of passengers on board of the buses. The resulting information is then processed in order to compute predictions for other buses, which are not equipped with the crowd-level sensing application. The resulting predictions are then fed back into the routing service such that they can be used (a) to guide routing decisions and (b) to inform the users.

Once a user has decided to follow a particular route proposal, the user can start a navigation session for the route. During this session, the GAMBAS middleware can automatically share the user's intended destination with the transit network operator. As explained later on, this allows the operator to detect routes that are going to be in high demand in the near future. Thereby, the user's identity is hidden from the operator. During navigation session, the user is supported through step-by-step instructions as shown in Figure 6.28.

The step-by-step instructions implement the concept of micro-navigation described in [FKR+14]. The idea behind micro-navigation is to optimally support the user's information needs during the usage of public transportation. For this, the app must provide the right pieces of information at the time when they are needed. To do this, the application usage text messages

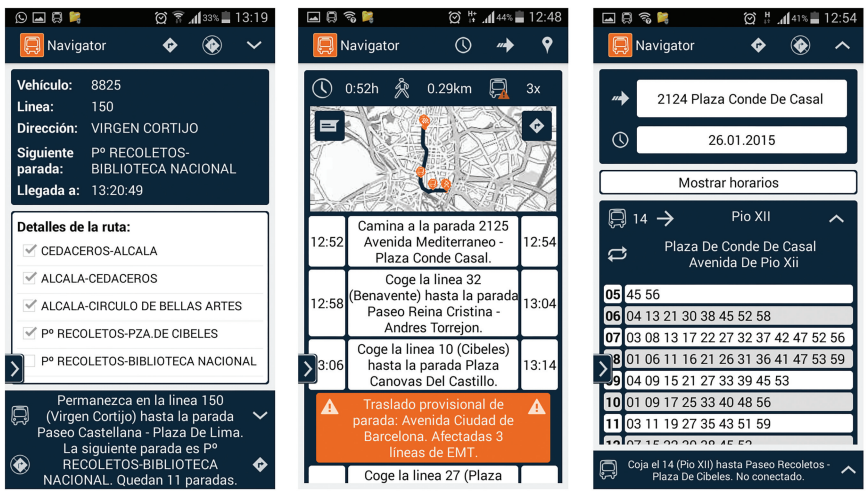


(a) (b) (c)

Figure 6.28 Madrid Navigator Navigation. (a) Walking, (b) Riding and (c) Textual.

that are shown at the bottom of the screen at all times. In addition, the application provides (optional) voice output using text-to-speech. To generate instructions, the application uses the GAMBAS data acquisition framework to tap into the sensors and information provided by the bus. To do this, the application automatically connects to the Wi-Fi network available in every bus operated by EMT Madrid and connects to the internal information system to determine the location and route of the bus. This information is then used to generate messages that correspond exactly to the user's context. For example, the app will notify the user to get off the bus shortly before it arrives at the correct stop. Similarly, if the user has taken the wrong bus, the app will immediately inform the user and propose a corrective action (e.g. to re-plan the route or to exit the bus at the next stop).

As shown in Figure 6.29, the app also enables users to directly access the bus information whenever they are traveling. This allows them to get real-time information about expected arrival times, even if they are not using micro-navigation. In addition, the application also integrates with the incident feed provided by the bus operator. This incident feed describes changes to schedules, e.g. due to demonstrations in the city center or traffic accidents. Thereby, the incidents are directly integrated into the routing results as well



(a) (b) (c)

Figure 6.29 Madrid Navigator Features. (a) Bus Infos, (b) Route Incidents and (c) Time Table.

as the timetable information that can be fetched for different stops. In addition to incidents, the timetable information also includes real-time information for buses that are departing within the next 20 minutes. To do this, the application integrates with a real-time service provided by EMT Madrid through the GAMBAS middleware.

6.3.3.2 Environmental Map

For the environmental scenario, we have developed a web-based application that enables end-users to inspect the state of the environment. This state is captured through measurements of pollutants that are acquired via the sensor deployment in buses and the noise-level measurements of the mobile noise-sensing application. For this, the environmental map application integrates with the GAMBAS noise service and the environmental service, described previously. After retrieving the data from them, the application applies the following data aggregation approach to create a visually appealing data representation:

1. Values corresponding to measurements at certain locations are clustered.
2. Based on the clusters, we identify the Voronoi partitions to define the area of the cluster.
3. Using Delaunay triangulation, we find adjacent areas to interpolate missing data.
4. Finally, we perform hexagonal binning in order to represent the result.

The resulting hexagonal visualization is then added to an overlay of tiles computed with the tile service, which results in the final result shown in Figure 6.30. Thus, using the web-based application, a user can simply move

(a) (b)

Figure 6.30 Environmental Map. (a) Voronoi Clusters and (b) Hexagonal Map.

the map to a specific location in the city and then view the different sensor readings in a manner that is easy to understand.

6.3.4 Operator Applications

In addition to the application services, sensing applications and mobile applications, we have also developed a number of applications that are not targeting the citizens. Instead, they are targeted towards the transit network operator, which, in our specific case, is EMT Madrid. The operator applications are aggregating the information collected through the crowd-level sensors and the mobile applications in order to help the operators to understand the current transit network usage. This understanding can then be used to optimize the network, possibly in real time, e.g. by dispatching additional buses or issuing route warnings, etc. In the following, we briefly outline these services.

6.3.4.1 Congestion Notifications

At the EMT Madrid headquarter, there are operators that control all the operations related to the bus network management. Crowd-level detection provides an estimation of the bus occupancy. A bus is considered as "congested" when a threshold of 85% of its capacity is exceeded. When the embedded application on the bus detects that a bus is getting congested, it generates a notification to signal this to the operator. If 2/3 of the vehicles within a route are congested, then the operator receives another notification that signals the congestion in the route. These alarms and notifications have been incorporated to the management system in a way that they can be visualized in the same graphical user interface that EMT is currently using. Figure 6.31 shows how the operator that is managing a route is notified when the threshold level is exceeded, by displaying an "Ocupacion LLENO" message (full occupation) and in red, the message "Ruta atocha-misericordia congestionada" (Atocha-misericordia route congested).

6.3.4.2 Demand Notifications

As described previously, the most demanded routes by the Madrid Navigator users are detected based on the usage of the navigation functionality. The currently used destinations during navigation are stored in the demand service. Once a certain number of destinations located in a certain area, for a given period of time, are reached, then that area can be categorized as a high-demanded destination zone. As a result, it will be shown to the bus

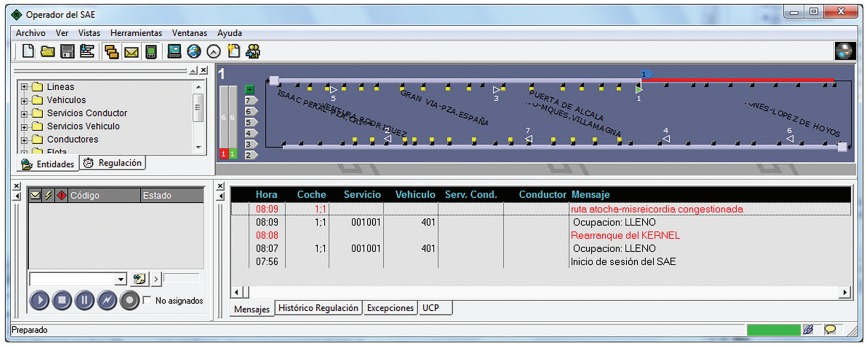
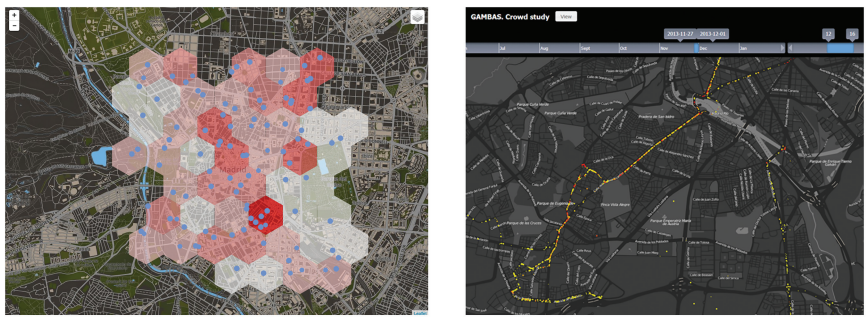


Figure 6.31 Congestion Notifications.



(a) (b)

Figure 6.32 Demand Notifcations and Occupancy Analysis. (a) Demand Notification and (b) Occupancy Analysis.

network operators who can use this information to detect a massive event such as a concert or a demonstration. Based on this, the operator can decide whether to reinforce the related bus lines covering that area or not. The information is offered to the bus operator in a map by using the hexagonal binning representation. The different hexagonal areas allow the operator to visualize the most demanded destinations in a quick and simple manner, as shown in Figure 6.32(a).

6.3.4.3 Occupancy Analysis

Crowd-level measurements are received in real time and stored in a data storage for offline analysis. For this storage service, we developed an operator tool to display the real-time and historical bus occupancy. Using this tool, the

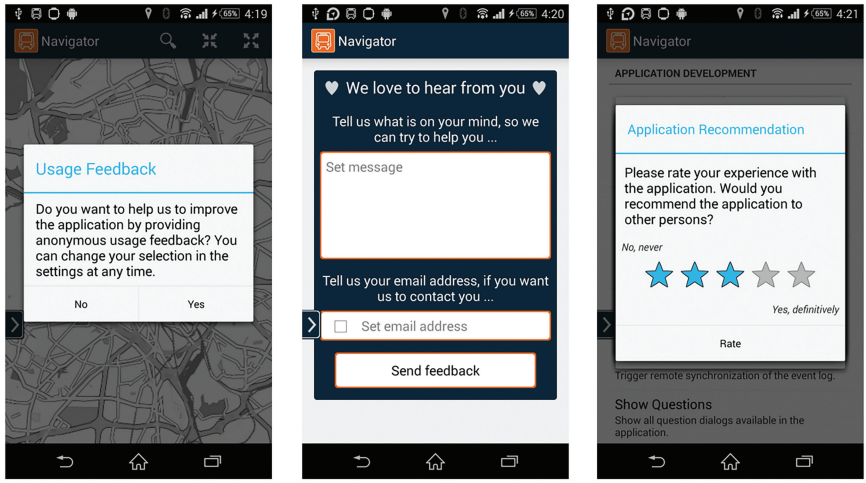
bus network operator is able to visualize occupancy information in a geo-located manner for a selected bus line. The viewer is implemented as a web application to visualize the buses location integrated with a map. The colors in the different routes are showing the crowd-level data at a specific time: low-crowded (green), medium-occupied (orange) and congested (red), in the same way as this information is shown in the mobile app.

6.4 Application Evaluation

During the course of the development of the GAMBAS middleware, we deployed all application services and sensing components. In addition, we performed a large-scale deployment of the Madrid Navigator navigation application. For the operator applications and the pollution map, we performed only internal testing with a closed user group. During the internal testing of the environmental applications, we found that the pollutant sensing system in the bus was not able to collect meaningful data. After an analysis and several rounds of discussions with the hardware manufacturer of the pollution sensor, we stopped the further roll-out of the system due to the unreliability of the sensor readings. As a consequence, the evaluation results described in the following are centered around the mobility scenario and the navigation app in particular.

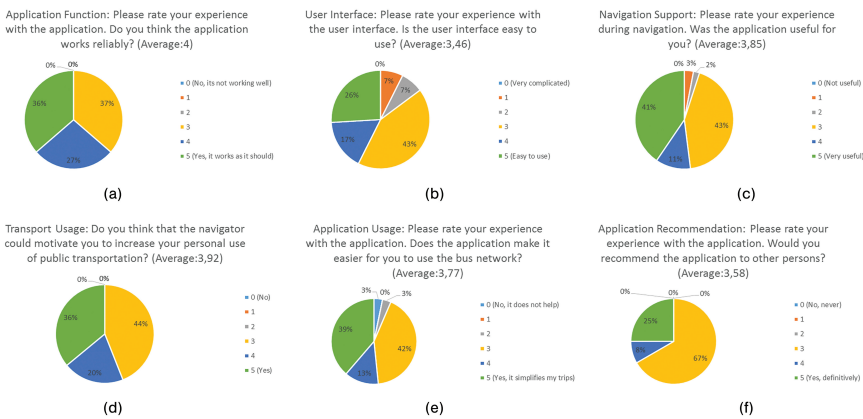
To evaluate the Madrid Navigator navigation app, we distributed it through the Android market in order to make it available to interested users and application developers. During the evaluation period, the application was downloaded more than 1000 times and used by both an internal group of testers and actual citizens that were not related to GAMBAS. From this deployment, we collected a significant amount of feedback both implicit (through the app usage) and explicit (through in-app questions and a feedback form). In the following, we briefly describe the application functionality and the results gathered during the deployment.

In order to detect issues and to improve the app during the deployment, we instrumented it with logging code. If a user gave his explicit consent as shown in Figure 6.33, we uploaded and analyzed the logs using the logging service described previously. In addition to implicit feedback, we also offered two ways to provide explicit feedback. First, we integrated a feedback form into the application and second, we used pop-up dialogs to ask user's about their current experience. For this, we implemented a regular 5-star rating dialog shown in Figure 6.33. Using the in-app questions, the users collectively generated 350 responses to different questions. Each of these questions could



(a) (b) (c)

Figure 6.33 Madrid Navigator Feedback. (a) Implicit, (b) Form and (c) Question.



(a) (b) (c)

(d) (e) (f)

Figure 6.34 Madrid Navigator Results. (a) Reliability, (b) Interface, (c) Navigation, (d) Motivation, (e) Usage and (f) Recommendation.

have been rated between 0 and 5 stars. The responses to each question are shown in Figure 6.34. In the following, we briefly discuss the results.

To determine whether the application worked as expected on the broad number of devices of the users, we asked the users to provide a rating with

respect to reliability. As depicted above, 36% of the users gave a 5-star rating (works as it should), 27% of the users gave a 4-star rating and 37% of the users gave a 3-star rating resulting in an average rating of 4 (out of 5). Consequently, we think that the mobile application was working well in many cases as none of the users gave a rating that was worse than 3 stars.

The second question that we posed to the users was to rate the overall usability of the user interface between easy-to-use (5 stars) and very complicated (0 stars). With 43%, the majority of users thought that the interface is neither easy nor complicated to use. Another 43% assigned a 4 or 5 star rating marking the interface clearly as easy-to-use. However, on the negative side, 14% of the users thought that the interface was rather complicated. We speculated that this could be due to issues on devices that have a small screen, which could result in usability issues with the map-based visualizations (e.g. small icons, etc.). However, we were not able to prove this assumption.

In addition to crowd-level and incident-aware routing, one of the core features of the GAMBAS Madrid Navigator is the application of context-awareness to enable intent-aware navigation instructions. Thus, in order to evaluate the usefulness of this feature, we asked the users whether they consider the navigation to be useful. Here, the overwhelming majority of users (95%) is rating the application with a 3 star or higher rating. 41% are rating the application even with the maximum rating resulting in an average of 3.85 stars. This clearly shows that a) the navigation was working reliable and b) the idea of micro-navigation was clearly considered to be useful.

In order to determine the impact of the Madrid Navigator on the user's transport behavior, we asked whether they think that the application could motivate them to use more public transportation. Here, the answer is again rather positive since 36% of the users completely agree that the application could motivate them and another 20% rather agrees, which results in an average rating of 3.92 stars. Consequently, we argue that navigation applications like the Madrid Navigator that employ context- and intent-awareness can be a benefit for transport network operators.

To determine whether the application actually helps users during their trips, we asked whether the application makes it easier for them to use the bus network. Again, the overall results were rather positive since 39% of the users stated that it would simplify their trips at least somewhat and only 6% answered that the application would not help. Thus, the overwhelming majority of users providing detailed and precise navigation instructions by means of context recognition at the right point in time can simplify their bus trips.

Finally, to gather the users overall impression on the Madrid Navigator, we asked them whether they would recommend the application to other users. Just like with previous questions on the reliability, usability and helpfulness of the application, the explicit user feedback reveals a rather positive result. With an average of 3.58 stars, the users are either undecided or would recommend the application.

In summary, these results are a clear indication for the maturity and usefulness of the navigation application. Given the fact that the implementation of the Madrid Navigator and all of its background services was leveraging the GAMBAS middleware, this also demonstrates the applicability of the abstractions provided by it. Thereby, it is important to stress that, in contrast to many other research projects, the tests were performed under realistic conditions with a large number of users that were not affiliated with the GAMBAS project.