3

Data Acquisition

This chapter describes the data acquisition framework of the GAMBAS middleware. The description includes discussions on the framework architecture, including the component system for developing context recognition applications and the activation system for enabling automatic, state-based activation of different configurations. The chapter also provides insight into the design rationale for the system. This includes a discussion of the motivation behind the component-based approach for context recognition, the chosen component model, energy-efficient techniques to perform context recognition on resource-constrained mobile devices, etc. Furthermore, rationale behind the state machine abstraction for the activation system and how energy optimization techniques used in the component system are fully utilized by the activation system is given. Before we discuss the framework, however, we first outline related work and clarify the innovations and research gaps closed by the data acquisition framework.

3.1 Focus and Contribution

Data acquisition is an essential part of any context recognition system. For such systems, data acquisition normally involves acquiring raw data from different types of sensors such as accelerometers, microphones, gyroscopes, proximity sensors, Wi-Fi, GPS, etc. The sensors can be embedded into a single device or alternatively, they can be embedded in different devices that are distributed in the environment. The data acquisition system acquires data from these sensors and pre-processes it before forwarding it to more complex recognition logic. Existing data acquisition systems differ depending on the leveraged resources and on the target application requirements. An efficient data acquisition system should be generic enough to be executable in different settings (different hardware and different application requirements) with little

or no tuning. In the following, we briefly review the state of the art for data acquisition systems mainly focusing personal mobile devices like smart mobile phones, PDAs, etc. Thereafter, we identify the gaps in the existing solutions and from these gaps, we derive a list of innovations realized by the data acquisition framework of the GAMBAS middleware.

3.1.1 Data Acquisition Frameworks

There exist a number of context data acquisition systems and frameworks for personal mobile devices [CK00]. These frameworks vary in their characteristics depending on their target applications and operating environments. Examples include [HH10], [DHH07], [BM10], [YTN05], [KZX+11], [LYL+10], [GJAS06], [RMM+10] and [CBSG12]. [HH10] describes a data acquisition framework for on-body sensor networks which runs on resource-constrained embedded systems and is used for human activity recognition. [DHH07] describes a context acquisition framework which allows the collection of raw sensor data from different sensing sources. The framework provides programming abstractions for developers to fetch data from different sensor implementation programs without developing the underlying communication mechanisms for the target platforms. [BM10] describes a service-oriented architecture based data acquisition framework. It allows sensor data fusion with local and external sources to build and manage context-aware services for personal mobile devices in a transparent manner. The framework protects the user's private data by using suitable privacy-preserving policies to handle information in P2P networks. [YTN05] describes a context acquisition framework based on a customized sensing platform named Muffin. Muffin supports a variety of sensors to help detect different types of contexts. The Citron framework running on Muffin uses a black box architecture for context processing and provides parallel processing of different sensor data streams (audio, accelerometer, etc.) to identify the user's context. [KZX+11] combines both on-body sensors and mobile phones for joint context recognition. The main contribution of this work is the provisioning of a framework to support the collaboration of TinyOS-based sensor modes and Android-based smart phones. This work also makes use of online training to improve the accuracy of the classifiers and it can automatically turn off redundant sensing sources to save energy. [LYL+10] describes a continuous sensing engine for context recognition applications. It uses the concept of pipes for different sensing sources (microphone, accelerometer, GPS) to balance out

the application requirements and the available resources. [GJAS06] outlines a software architecture and a service for on-body sensors as part of the user's attire. The system is realized using MicaZ motes. Challenges addressed in this work include storage of data, uploading of data, synchronization of data, power management of motes, reconstruction of activity logs, user interfaces, etc. The presented architecture is aimed at the future development of smart attire systems. [RMM+10] is a data acquisition framework for detecting user's social and physiological patterns using smart mobile phones. The system can be programmed using a declarative language to describe user behavior models, action base and knowledge base. The system can be adapted at runtime to activate and deactivate sensors. The recognition is based on GMMs (Gaussian mixture models). The system is aimed at helping social scientist to understand the correlation of user emotions with the places, groups and their activities. [CBSG12] is a collaborative context recognition system for smart mobile phones. The system execution is a two-stage process consisting of stages, namely grouping stage and context recognition stage. In the grouping stage, devices are clustered based on their proximity. Once devices are clustered, they scan the environment and send the raw data for subsequent context recognition to a backend server. In the context recognition stage, the system uses coupled hidden Markov models to model activity and location sequences. The system is aimed at advertisement systems where advertisements are shown based on mutual context and interest of user groups.

3.1.2 Rapid Prototyping Tools

There also exist a number of rapid prototyping tools for expeditious development of context recognition applications. Commonly known tools include [SDA99], [BAL08] and [TRL+09]. [SDA99] is targeted at context recognition with pre-deployed sensors and provides a uniform abstraction for applications to access and use context information by hiding the actual context sensing and interpretation from applications. [BAL08] is targeted towards activity recognition for wearable systems. This toolkit provides functionalities to develop distributed context recognition systems as well as reusable components, parametrizable algorithms, filters and classifiers. [TRL+09] is a data gathering and processing open-source platform targeted towards mobile phones with varying hardware capabilities. It consists of a minimal core that can be extended by plug-ins.

3.1.3 Application-Specific Acquisition

The systems mentioned above are generally used for dealing with heterogeneous sensing sources and providing flexibility for application developers to customize applications in a certain way. However, there exist a number of fine-tuned data acquisition and context recognition systems that can only be used in a narrow set of situations. Examples include [BC09], [MLEC07], [LLEC08], [LPL+09] and [EML+07]. These and many alike systems are manually fine-tuned for particular applications and therefore are able to detect only the fixed set of characteristics. As a result, these systems cannot be adapted to dynamic environments which a user might experience in a daily routine. They use built-in sensors in mobile phones to recognize the required context. For instance, [MLEC07] uses microphones and accelerometers to determine user context which is then injected into social networking websites. [LLEC08] uses accelerometers and microphones to detect road conditions. [TRL+09] uses location sensors to identify road traffic congestions. Sound Sense [LPL+09] uses a microphone to classify different types of sounds in the surrounding. [BC09] is a system aimed at video recording of social events in a distributed manner using mobile phones. The phones are grouped based on the social activity in which their users are involved. For detection of a social activity, a phone at the appropriate location is chosen to record events. At the end of the social activity, all recordings from different phones are combined into one video by a backend server to create a video highlighting important events of the social gathering.

Data acquisition and retrieval of contextual information is a resource consuming process which can have a significant effect on overall system performance for resource constrained personal mobile devices. Over the last years, there has been some work towards devising mechanisms for achieving energy-efficient data acquisition and processing. Examples include [KLJ+08], [WLA+09], [RH10] and [RMJ+11]. [KLJ+08] detects changes in the context data at an early stage. For instance, rather than waiting for the results from the classifier, the system detects changes in sample values at the sensor level. Thereafter, only those samples are further processed which can lead in a context change, whereas [WLA+09] uses hierarchical sensor management strategy to detect user states and state transitions and only fires a transition when a particular transition probability is met. As a result, this reduces the unnecessary execution of unwanted sensors. [RMJ+11] is aimed at computing multiple contexts from multiple sensing sources. The authors have proposed a theoretical model that shows the inaccuracy of estimating

multiple contexts from multiple sensing sources. The work also presents a heuristic algorithm for searching the set of sensors to recognize the required multiple contexts.

3.1.4 Contribution

Designing a context data acquisition system is usually driven by the target applications and operating environments. Therefore, such systems are optimized with considerations to their requirements. The above-mentioned systems are similarly aimed at optimizing a particular characteristic, which could be the efficient utilization of available resources or the highly accurate recognition of a particular context or the efficient prototyping of context recognition applications. Looking at the description of these systems reveals a need for a generic yet efficient system that in essence should be a complete framework, which, on the one hand, allows efficient usage of available resource and, on the other hand, supports rapid development of specialized recognition applications with high accuracy. The data acquisition framework in GAMBAS middleware bridges these gaps. It aims at providing a complete solution that meets all the aforementioned objectives. Specifically, the data acquisition framework of the GAMBAS middleware adopts a component-based approach allowing multi-modal context data acquisition. The framework provides an extensive component toolkit for rapid development of new context recognition tasks. Using a component-based solution, the data acquisition framework applies resource-efficient techniques (memory, energy, etc.) with no or little impact on the recognition accuracy. Moreover, the data acquisition framework is executable in distributed settings to enhance the quality of desired context and helps in providing relevant services to different groups of users (depending on their location, interests, etc.). Finally, the framework provides a number of basic components that can be used to build applications. These components cover activity and intent recognition as well as sound and speech recogntion.

The activity recognition components in the data acquisition framework focus on computing various user activities or user contexts. Due to the application scenarios targeted by the GAMBAS middleware, the primary focus lies on location-based activities, e.g. shopping in a supermarket, waiting for the bus at the stop, traveling in a bus (standing or sitting), sightseeing in a new city, etc. The data acquisition components rely on a variety of means (motion sensor, Wi-Fi, GPS, on-line calendars) to recognize these and similar activities. Similarly, the data acquisition framework encompasses necessary

components to estimate the user's intents. By this, we mean the user's likely location or activity in the future, e.g. knowing that a user is traveling on a bus to his destination, it might be useful or interesting to notify him about the possibility of meeting a friend. If he is willing to change his route, then prompt him of new shopping facilities near the destination. The intent recognition components support computing such user intents based on user's activity patterns or interests.

The sound and speech recognition components focus on interpreting acoustic signals in the environment of the user. A primary focus lies on the recognition of environmental sounds, like engine sounds, traffic noise, talking people, etc. to determine the means of transportation. The goal is to identify delays in public transportation to adjust the predictions of personal intentions. The so acquired data can be distributed in accordance with the privacy setting to optimize travel plans of other users who rely on the same means of transportation. The components use historic data and compare it to live data to identify differences in schedule or behavior patterns. The speech recognition components are designed to allow the integration into other applications on the device. This allows developers to create new applications that offer voice control via speech recognition.

3.2 Data Acquisition Framework

The data acquisition framework (DQF) is one of the fundamental building blocks of the GAMBAS middleware. Conceptually, the DQF is responsible for context recognition on personal mobile devices including smart phones, PDAs and laptops. The DQF supports various platforms including Android, Windows and Linux. It is realized as a multi-stage system. At lower stages, it allows developing reusable components and component compositions for context recognition applications. At higher stages, it enables application developers to automatically activate compositions when needed. To do this, the DQF is split into two parts as shown in Figure 3.1, a component system and an activation system.

The component system uses a component abstraction to enable the composition of different context recognition stacks that are executed continuously. A context recognition stack or simply a configuration refers to a set of sampling, preprocessing and classification components wired together to detect a specific context. Examples of such contexts include the physical activity of a person, the location of a person, etc. The configurations can be used to detect context for a multitude of purposes and have applications in

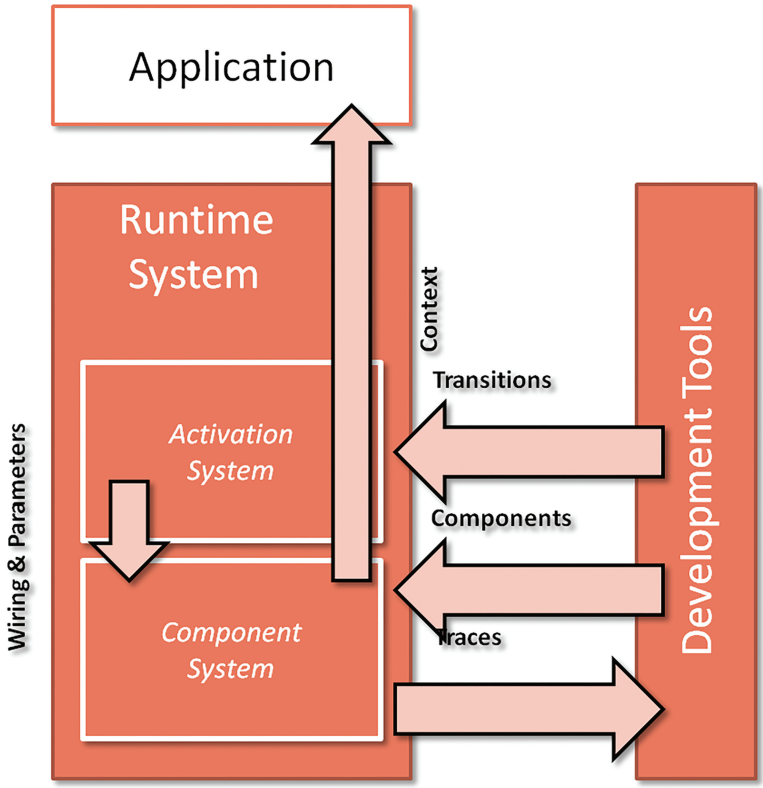

Figure 3.1 Data Acquisition Framework Overview.

areas of smart home environments, assisted living for elderly, proactive route planning, shopping, etc.

The activation system uses a state machine abstraction to determine the point in time when a certain configuration or a set of configurations should be enabled. The activation system enables the required configurations in an automatic manner based on the conditions associated with the state transitions. An example of a simple (coarsely granular) state machine associated with an employee could consist of two states, "Working" and "Relaxing". State "Working" may consist of configurations "Meeting", "Cafeteria", etc. and state "Relaxing" may consist of configurations "Living Room" and "Gardening". Based on the transition values, the activation system will disable the configurations associated with one and enable the ones associated with the other. In addition, the state machines can also have more fine granular

states representing stages specific to a single task, e.g. a state can represent the sampling of an accelerometer with lower or higher rate. In such a case, a state change may occur when the device screen turns on, for instance. In the following, we describe both systems in detail.

3.2.1 Component System

At the lower level of the data acquisition framework, context and activity recognition is done using a component-based approach which promotes reusability and rapid prototyping. In addition, this approach also enables the automated analysis of application structures in order to optimize their execution with respect to energy efficiency.

From the perspective of the component system, each application consists of two parts: the part containing the recognition logic and the part containing the remaining application logic. The part that contains the recognition logic usually consists of sampling, preprocessing and classification components that are connected in a specific manner as shown in Figure 3.2. The part that contains the remaining application logic can be structured arbitrarily. Upon start up, a context recognition application passes the required configuration to the component system, which then instantiates the specified components and executes them. Upon closing, the configuration is removed by the component system which eventually releases the components that are no longer required. The component system is implemented in Java and supports various platforms

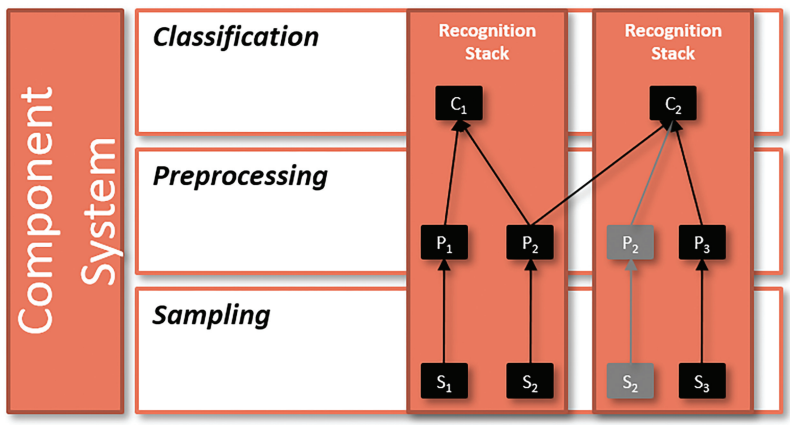

Figure 3.2 Component System Overview.

including J2SE environments and Android. Using an Eclipse-based graphical editor, application developers can visually create configurations by selecting and parameterizing components and by wiring them as needed. In the following, we first provide more details on the underlying component model, before we discuss the runtime and development support.

3.2.1.1 Component Model

To structure the recognition logic, the component system realizes a light-weight component model which introduces three abstractions. First, *components* represent different operations at a developer-defined level of granularity. Second, *connectors* are used to represent both the data and the control flow between individual components. Third, *configurations* are used to define a particular composition of components that recognizes one or more context characteristics.

3.2.1.1.1 *Components*

Components are atomic, reusable building blocks that constitute the recognition logic. Similar to other systems such as J2EE or OSGi, components can be defined at arbitrary levels of granularity. Yet, in contrast, they can be instantiated multiple times and they are parameterizable to support different application requirements. Due to the support for parametrization, the component model is more flexible than other models. In addition to parameters, all components exhibit a simple life cycle that consists of a started and a stopped state. To interact with other components, a component may declare a set of typed input and output ports that can be connected to other components using connectors.

As depicted in Figure 3.3, the recognition logic of a speech detection application may, for example, consist of a number of components which can be divided into three levels. At the lowest level, the sampling components are used to gather raw data from an audio sensor. On top of the sampling components, a set of preprocessing components take care of various transformations, noise removal and feature extraction. Finally, the extracted features are fed into (a hierarchy of) classifier components that detect the desired characteristics. Depending on the purpose and extent of the application logic, it is usually possible to further subdivide the layers into smaller operators. Although the component system does not enforce a particular granularity, such operators should usually be implemented as individual components to maximize the potential for reuse.

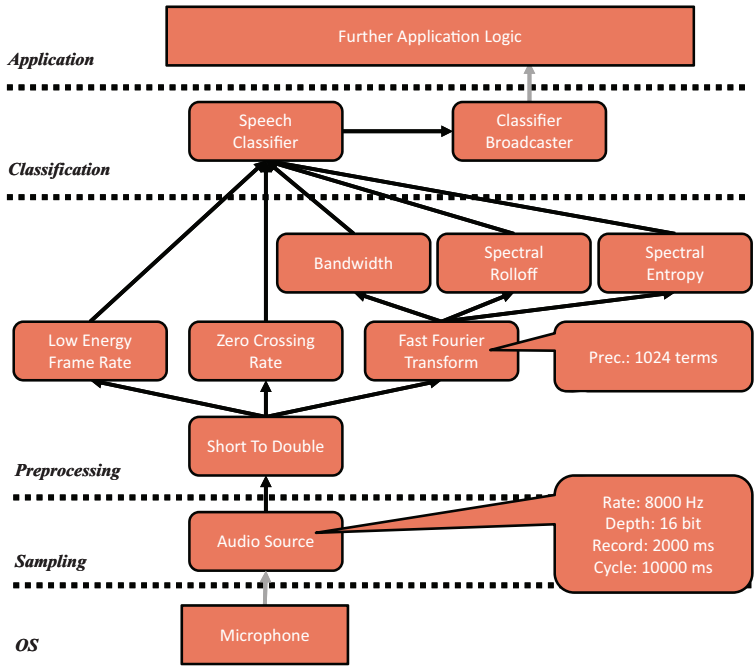

Figure 3.3 Speech Detection Configuration Example.

3.2.1.1.2 *Parameters*

Parameterizations increase the reusability of a component implementation across different applications. The component system allows components to support a developer-defined set of parameters. Components expose these parameters to adapt their internal behavior. As shown in Figure 3.3, at the sampling layer, these parameters might be used to express different sampling rates, sampling depths, frame sizes and duty cycles. At the preprocessing layer, they might be used to configure different filters or the precision of a transformation. In the component system, the parameters are not exposed to other components. Instead, they can be accessed and manipulated by the components.

3.2.1.1.3 *Ports*

In order to support application-independent composition, each component may declare a number of strongly typed input and output ports. Input ports are used to access results from other components. Output ports are used to

transfer computed results to another component. Thus, ports enable components to interact with each other in a controlled manner. The developer can add multiple input and output ports of different types. The component system takes care of the necessary memory allocation and de-allocation and performs efficient buffer management for each of the ports in transparent manner.

3.2.1.1.4 *Connectors*

In order to be reusable, components are isolated from each other by means of ports. However, the recognition of a context feature often requires the combination of multiple components in a specific way. Connectors express such combinations by determining how the typed input and output ports of different components are connected with each other. In order to minimize the overhead of the component abstraction, connectors are implemented using an observer pattern [GHJV95] in which the output ports are acting as subjects, whereas the input ports are acting as observers. This enables modeling of 1:n relationships between the components, which is required to avoid duplicate computations. To avoid strong coupling between components, input ports do not register themselves at the output ports, but the component system takes care of managing all required connections. An example of connectors can be seen in Figure 3.3, where the output port of the fast Fourier transform component is connected to the input ports of the bandwidth, the spectral roll off and the spectral entropy component.

3.2.1.1.5 *Configurations*

To recognize a particular piece of context, a context recognition application must explicitly list all required components together with their connectors in a so-called configuration. While this approach slightly increases the development effort, it also increases the potential reuse of components that can be applied on data coming from different sources. As an example of such component, consider a Fast Fourier Transform (FFT) that converts a signal from its time domain into the frequency domain. Clearly, such a component can be applied to various types of signals such as acceleration measurements or audio signals. Thus, by explicitly modeling the wiring of components as part of a configuration, it is possible to reuse this component in different application contexts. In addition to listing components together with their connectors, the support for parameterizable components also requires the developer to explicitly specify a complete set of parameter values that shall be used by each component. As a result, every configuration consists of a

parameterization as well as associated connectors. An example of a speech detection configuration is shown in Figure 3.3.

3.2.1.2 Runtime System

The main task of the runtime system of the component system is to support the execution of configurations defined by different context recognition applications in an energy-efficient manner. This includes loading the configurations specified by the context recognition applications, instantiating the components with right parameterizations and connecting them in the manner specified by the application. In addition to that, the runtime system applies energy optimization techniques if more than one application is executed simultaneously. When the applications do not require the context information anymore, the runtime system stops executing the associated configurations. A detailed description of the component system structure and execution of applications is given in the following sections.

3.2.1.2.1 *System Structure*

As shown in Figure 3.4, the main elements of the runtime system of the component system are the configuration store, the configuration folding algorithm [IHW+12] and the applications. The configuration store is used to cache the configurations associated with applications that are active. It is also used to store their folded configuration. The configuration folding algorithm provides energy-efficient execution of context recognition applications, provided that more than one application is executed simultaneously. The entity responsible for managing the runtime system is called the component manager.

3.2.1.2.2 *Configuration Execution*

The component manager controls the execution of the componentized recognition logic of all running applications. To manipulate the components executed at any point in time, the component manager provides an API that enables developers to add and remove configurations at runtime. When a new configuration is added, the component manager first stores the configuration internally. Then, it initiates a reconfiguration of the running recognition logic that reflects the modified set of required configurations. To reduce the energy requirements, the component manager does not directly start the components contained in the configuration. Instead, it uses the set of active configurations as an input for our configuration folding algorithm.

The goal of the configuration folding algorithm is to remove redundant components that are present in different applications and perform the same

sampling or compute redundant results. Using the set of configurations, the configuration folding algorithm computes a single, folded configuration that produces all results required by all running applications without duplicate sampling or computation. Once the configuration has been folded, the component manager forwards it to the delta configuration activator. By comparing the running and the folded configuration, the activator determines and executes the set of life cycle and connection management operations (starting, stopping and rewiring of components) that must be applied to the running configuration in order to transform it into the folded target configuration. When executing the different operations, the delta activator takes care of ensuring that their ordering adheres to the guarantees provided by the component life cycle. To do this, it stops existing components before they are manipulated. This procedure is illustrated in Figure 3.4.

3.2.1.2.3 *Platform Support*

The core abstractions of the component systems as well as the component manager are implemented in Java 1.5. In order to support multiple platforms, different wrappers have been implemented that simplify the usage of the component system on platforms including Windows, Linux and Android.

3.2.1.3 Tool Support

The component system encompasses offline tools to support rapid prototyping. These tools include a visual editor which is used for creating and

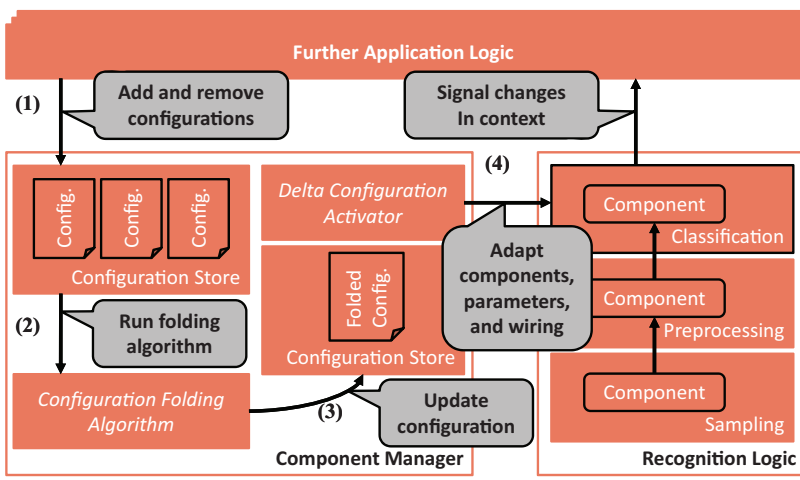

Figure 3.4 Component System Structure.

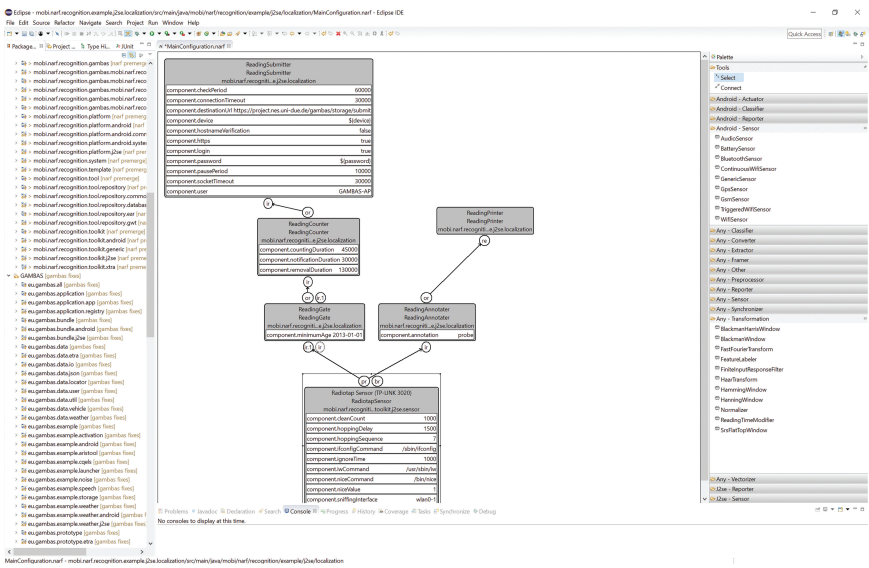

Figure 3.5 Component System Tool Support.

updating configurations for the context recognition applications. The visual editor provides a user-friendly interface, which allows developers to drag, drop, parameterize and wire existing components to create new configurations or update existing ones. The visual editor is implemented as a plug-in for the Eclipse IDE (Version 3.7 and above). A screenshot of the visual editor is shown in Figure 3.5.

In addition to the visual editor, the component system also provides a large set of generic sampling, preprocessing and classification components as part of the component toolkit. At the sampling level, the toolkit provides components that access sensors available on most personal mobile devices. This includes physical sensors such as accelerometers, microphones, magnetometers, GPS as well as Wi-Fi and Bluetooth scanning. In addition, the toolkit encompasses components that provide access to virtual sensors, for instance, personal calendars.

For preprocessing, the toolkit contains various components for signal processing and statistical analysis. This includes simple components that compute averages, percentiles, variances, entropies, etc. over data frames as well as more complicated components such as finite impulse response filters, fast Fourier transformations, gates, etc. Furthermore, the toolkit also contains

a number of specialized feature extraction components that compute features for different types of sensors such as the spectral rolloff and entropy or zero crossing rate, which are used in audio recognition applications [LPL+09] or Wi-Fi fingerprints, which can be used for indoor localization.

At the classification layer, the toolkit contains a number of trained classifiers, which we created as part of the audio and motion recognition applications. Finally, there are a number of platform-specific components which are used to forward context to an application which enables the development of platform-independent classifiers. On Android, for example, a developer can attach the output of a classifier to a broadcast component which sends results to interested applications using broadcast intents. We have also developed a number of components that are useful for application development and performance evaluation. These includes components that record raw data streams coming from sensors as well as pseudo sensors that generate readings using pre-recorded data streams. Together, these components can greatly simplify the application development process on mobile devices as they enable the emulation of sensors that might not be available on a development machine.

3.2.2 Activation System

To fully understand the context of a person, it is usually necessary to recognize more than one context characteristic. As an example, consider that to know if a person is working in his office, context characteristics such as his location, pattern of movement, types of meetings and classification of ambient sounds are required. As described earlier, such context characteristics can be detected using the component system by developing configurations with the appropriate components, parameterizations and connections. Furthermore, in order to fully identify a particular context, more than one configuration would be needed at a particular time. In real life, however, the context of an entity does not remain static and over the period of time, it requires detection of different context characteristics.

Moreover, the context of a person depends on the task that the person is involved in. In other words, to know the context of a person, it is essential to know the current task. Furthermore, these tasks often follow certain patterns, e.g. tasks that a working person usually has consist of waking up in the morning, dressing up according to the weather, traveling to the work place, sitting in the office, holding meetings and discussions, going for lunch and coffee breaks, working on a computer, going for shopping, going home,

relaxing, having dinner, sleeping, etc. Thus, the resulting routine is often predictable, at least partially.

Given the presence of such regular patterns of reoccurring tasks, the goal of the activation system is to exploit the knowledge about their existence in order to minimize the amount of sampling and processing that is needed to detect the user's context. To do this, the activation system enables the developer to model individual tasks as a set of states that occur sequentially. For each of the states, the developer may specify a set of configurations that describe the context that shall be recognized. In addition, the developer specifies a set of transitions between the states that define possible sequences. Using this model, the activation system takes care of executing the right configurations at the right time as shown in Figure 3.6. In the following, we describe this basic idea in more detail.

3.2.2.1 Activation Model

In the GAMBAS data acquisition framework, the modeling of the routines of a task is supported by the activation system, which uses a state machine as its primary model. Specifically, the activation system enables the automatic, state-based activation of different configurations associated with developer-defined tasks. Hence, in the activation system, the entity's context is modeled as a state with different configurations associated with it, irrespective of its granularity. The transitions between the states are modeled using context-dependent rules. In the following, we discuss these concepts in more detail.

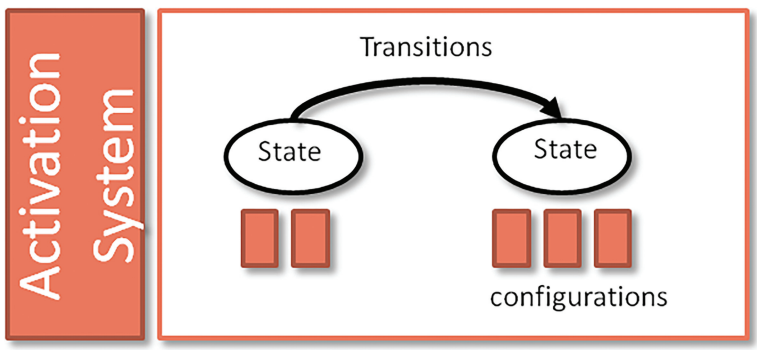

Figure 3.6 Activation System Overview.

3.2.2.1.1 *States*

A state refers to a particular decision point during the execution of a larger task. It entails a set of configurations that individually detect different context characteristics but collectively identify one of the possible decisions taken by the user.

For this purpose, states may be used to model decision points at different levels of granularity. An example of a coarse-grained state is shown in Figure 3.7(a). In this example, a high-level "working" state may encompass configurations that detect whether the person is in a meeting, working in his office or having lunch at the canteen. An example for a fine-grained use of state is shown in Figure 3.7(b). Here, the state "Fast Sampling" may be used in conjunction with a "Slow Sampling" state in order to control the precision of a certain set of configurations such as a movement detector or a sound classifier.

3.2.2.1.2 *Transitions*

Transitions are defined by the conditional changes in the configurations associated with a state. When the changes suggest that a certain condition holds, the activation systems disables the current state and its associated

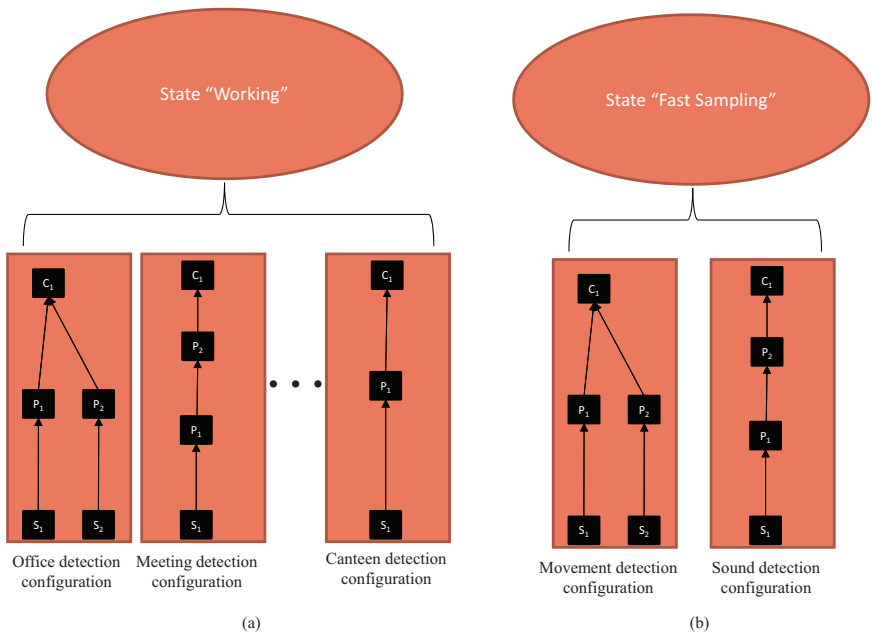

Figure 3.7 Examples of Activation System States. (a) Coarse-grained Usage and (b) Fine-grained Usage.

configurations and enables the ones associated with the new state. The activation system uses rules to model the conditions. Internally, each rule is represented by an abstract syntax tree, in which expressions for each configuration are defined. Depending on the evaluation of the expressions, the activation system decides whether a state must be changed.

Figure 3.8(a) shows two example states. State 1 has two configurations, Configuration A and Configuration B. State 2 also has two configurations, Configuration C and Configuration D. The transition from State 1 to State 2 is labeled as Transition $1 \rightarrow 2$, and the transition from State 2 to State 1 is labeled as Transition $2 \rightarrow 1$.

The abstract syntax tree of the rule for Transition $1 \rightarrow 2$ and Transition $2 \rightarrow 1$ is shown in Figure 3.8(b) and Figure 3.8(c), respectively. Assuming that State 1 is currently the active state, the activation system continuously evaluates the rules defined by the expression of Transition $1 \rightarrow 2$ and when the outcome of the expression, here represented by an AND operator, is true, it will disable Configuration A and Configuration B and enable Configuration C and Configuration D. Similarly, when State 2 is the current state, the activation system evaluates the rules associated with

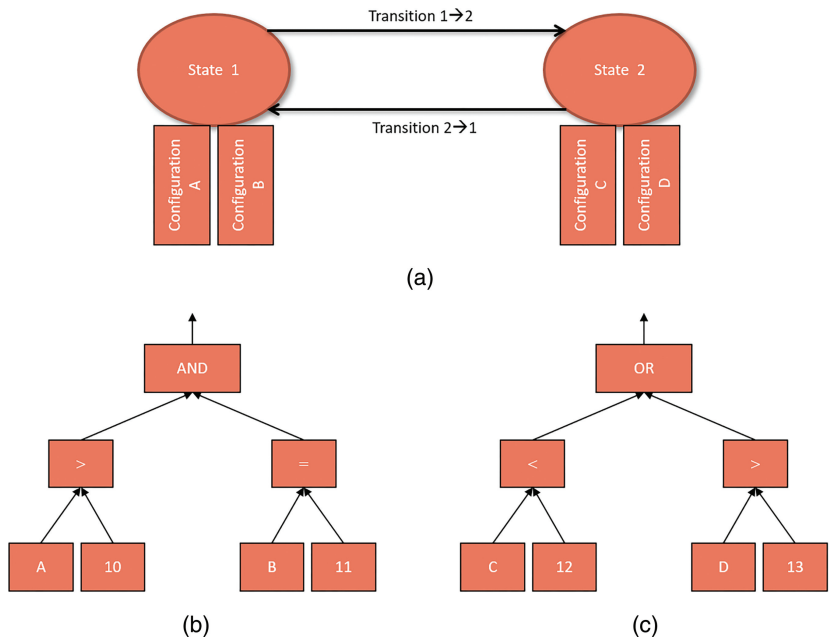

Figure 3.8 Examples of Activation System Transitions. (a) Activation System Transition Example, (b) Transition from 1 to 2 and (c) Transition from 2 to 1.

Transition $2 \rightarrow 1$ and it will execute the associated state change whenever this is implied by the outcome.

3.2.2.2 Runtime System

The main task of the runtime system is to load and execute the state machines defined by different applications. For this, the system instantiates the configurations associated with states, identifies the current state, instantiates rules for different transitions and evaluates the expressions associated with the respective transitions. Thereby, the activation system executes the state machines in an energy-efficient manner by applying configuration folding among all configurations across all the different states. The outcome of such a "folded" state machine is a single-folded configuration. Clearly, it is possible that in such a folded configuration, different configurations share the same graph structure, at least to a certain level. Therefore, the activation system provides logic for evaluating transition between the states.

3.2.2.2.1 *System Structure*

The main structural elements of the activation system are shown in Figure 3.9. These include a state machine store, the configuration folding algorithm, a rule engine and the state machine manager. The state machine store is used to

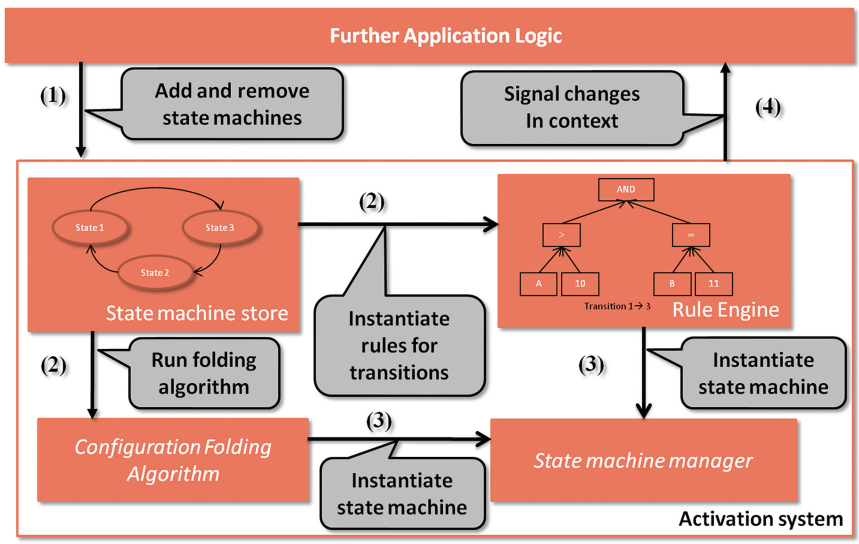

Figure 3.9 Activation System Structure.

cache the state machines associated with the applications. The configuration folding algorithm is used to compute an energy-efficient configuration for an entire state machine. To do this, the activation system applies configuration folding on the configurations of the currently executed state machines. The transitions between the states are modeled as if-else conditions and are managed by the rule engine. Once the folded configuration of the state machine and the rules for the state transitions are loaded, the state machine manager attaches the rules in the folded configuration, instantiates it and executes it. Similar to the component system, when the application logic indicates that no further context information is needed, the activation system stops executing the state machine.

3.2.2.2.2 *Configuration Mapping*

To provide a better understanding of the integration between the component system and the activation system, we describe how the configurations related to different states are folded and how the rule engine applies rules representing transitions between the states. To understand the mapping, consider an example of a state machine with two states as shown in Figure 3.10(a). Each state has two configurations attached to it. When the activation system loads the state machine, it applies the configuration folding algorithm on all configurations associated with both states, and the result is shown in Figure 3.10(b).

Let us assume that the rules for the two transitions are defined as follows:

- $1 \rightarrow 2$**: IF** Config. A **OR** Config. B **EQUALS** false **THEN** State 2
- $2 \rightarrow 1$**: IF** Config. C **OR** Config. D **EQUALS** false **THEN** State 1

The resulting mapping for the states, transitions and the folded configurations of State 1 and State 2 are shown in Figure 3.11(a) and Figure 3.11(b), respectively. If the state machine is residing in State 1 (c.f. Figure 3.11(a)), the configurations that must be evaluated according to the definition are Configuration A and Configuration B. Since folding has already taken place for all configurations of the state machine, the required graph structure for Configurations A and B is distributed across in two different graphs. However, these graph structures also share configurations from other states. Therefore, in order to evaluate the relevant configurations only, the activation system enables only the components that are required to compute Configuration A and Configuration B as shown in Figure 3.11(a). The remaining components are disabled. During the execution of the components required for State 1,

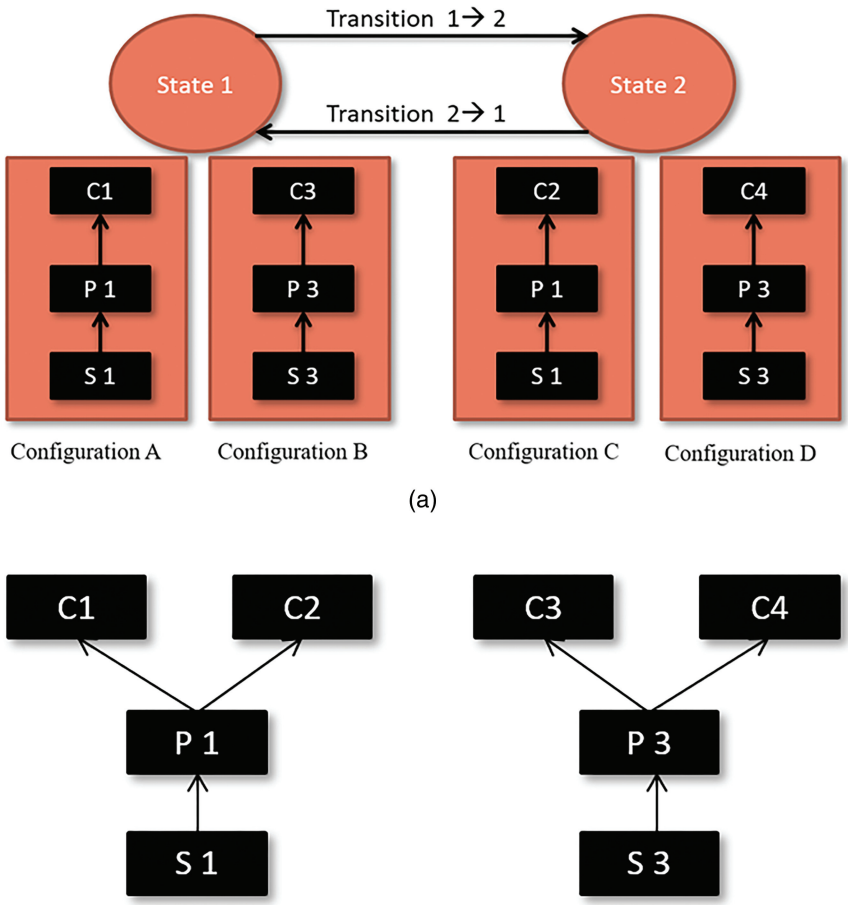

Figure 3.10 Configuration Mapping Example. (a) States, Transitions and Configurations and (b) Resulting Folded Configurations.

the activation system continuously evaluates the rule for the transition from State 1 to State 2 using the rule's syntax tree.

When the conditions defined by one of the active rules hold, the activation system initiates the state transition. Thereby, it stops the configurations of the previous state that are no longer needed and it starts the configurations required by the new state. In addition, the system stops the evaluation of the rules associated with the previous state and begins with the evaluation of the

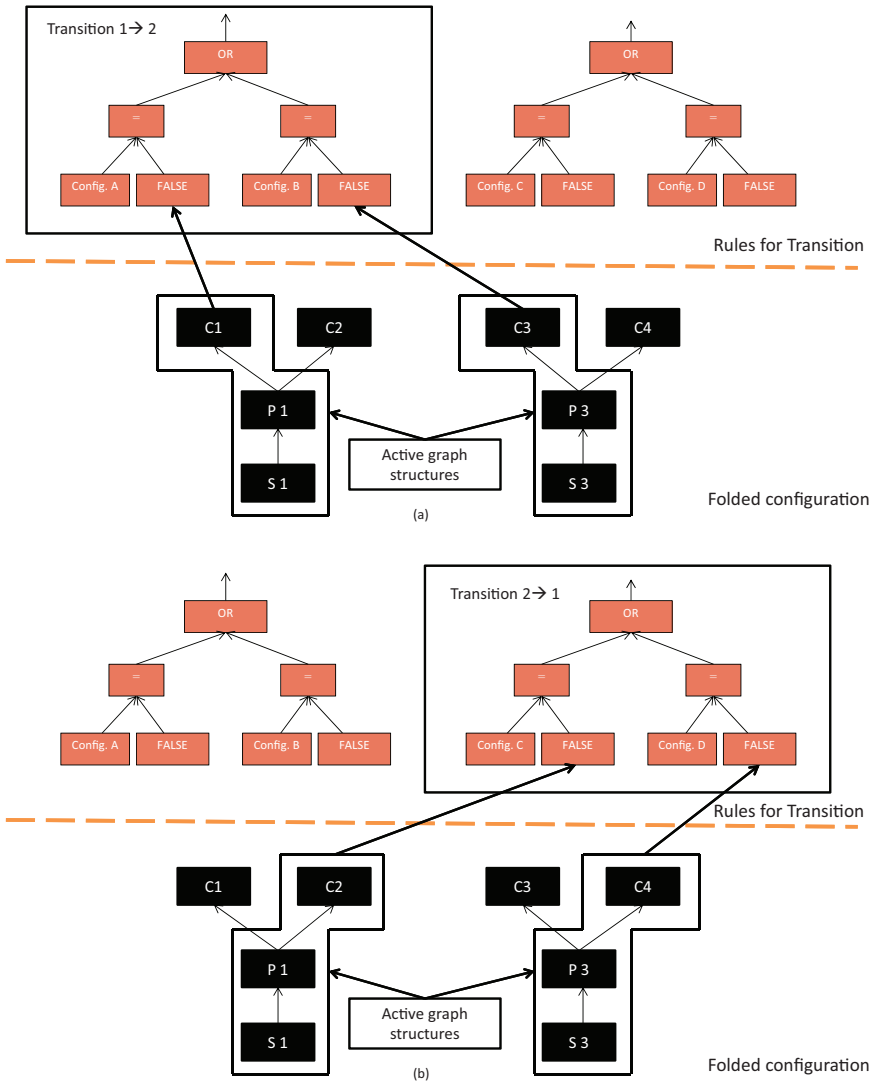

Figure 3.11 Executed Configurations and Transitions. (a) State 1 and (b) State 2.

rules for the new state. The result after transitioning from State 1 to State 2 is shown in Figure 3.11(b). Once State 2 becomes active, the system activates the Configurations C and D which are associated with State 2 and it begins the evaluation of the transition rule from State 2 to State 1.

3.2.2.2.3 *Platform Support*

Similar to the component system described previously, the core abstractions of the activation systems have been implemented using Java 1.5. In order to support multiple platforms, different wrappers have been implemented that simplify the usage of the activation system on platforms including Windows, Linux and Android.

3.2.2.2.4 *Tool Support*

Just like the component system, the activation system also provides a suite of offline tools to support rapid prototyping. These tools include a visual editor which simplifies the definition of states and transitions. The visual editor provides a user-friendly interface which allows developers to drag, drop, parameterize and wire existing configurations to create new state machines or to update existing ones. Similar to the visual editor of the component system, the visual editor for the activation system is also implemented as a plug-in for the widely used Eclipse IDE.

In addition to the visual editor, the activation system provides a set of configurations as part of the configuration toolkit for detecting different context such as location, speech, motion, etc. With the availability of the toolkit, developers do not have to create configurations from scratch. Instead, they can reuse existing configurations with trained classifiers, which can significantly reduce the application development time. A screenshot of the tool support for component system is shown in Figure 3.12.

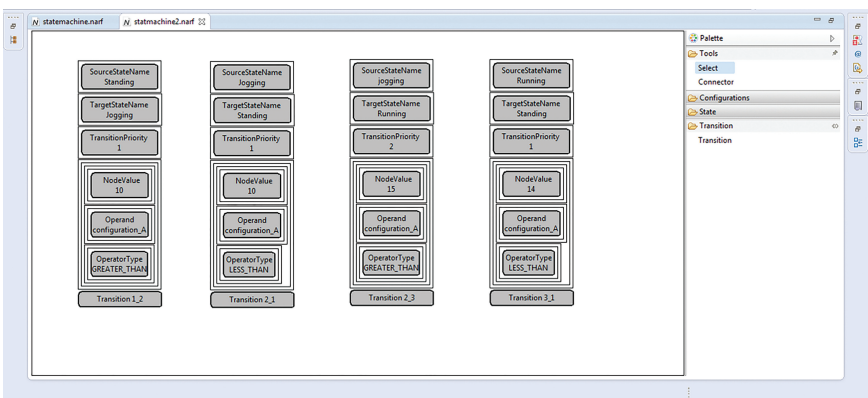

Figure 3.12 Activation System Tool Support.

3.3 Data Acquisition Components

As indicated by the previous discussions, the data acquisition framework of the GAMBAS middleware is highly configurable and extensible to support the acquisition and processing of arbitrary data from different sources. Using component compositions and state-machine definitions, even complex context recognition tasks can be supported in a highly structured manner. In order to speed up the development of applications, the data acquisition framework contains a set of basic recognition stacks including (trained) classifiers that support a broad variety of low-level and high-level context acquisition tasks. Using these building blocks, we have realized a broad number of applications described in more detail in Chapter 6. However, since they are usable beyond the scope of these applications, we briefly describe them in the following.

3.3.1 Context Recognition

The context recognition components are the basic building blocks of a context recognition application. The component toolkit provided with the component system consists of a large number of sampling, preprocessing and classification components. These components can be used to create new applications. Moreover, with the help of the toolkit, developers can implement their own components with little effort. Due to the targeted application scenarios described in Section 1.3, the components that we developed with the GAMBAS middlware are primarily focusing on location recognition, trip recognition and sound recognition.

3.3.1.1 Location Recognition

In order to determine the location of the user, the location recognition components integrate GPS information with RF signals that are present in the user's environment. Specifically, the components combine information from GPS, GSM and Wi-Fi sensors of the user's phone. Each of them has its own advantages and limitations but their collective use can provide efficient and accurate location recognition. With the widespread use of Wi-Fi, a user can typically see multiple Wi-Fi access points in the surroundings. With the limited range or signal strength of a typical Wi-Fi signal, a user can see different set of access points as he moves from one location to another. Thus, capturing this information alone can provide the user with a good view of his location. However, in places where Wi-Fi signals are not available or are very weak, GSM signals can be used instead. Typically a mobile phone can

report up to 6 neighboring cell towers. Though the range of a GSM cell tower is usually large and same locations may exhibit identical cell information, together with Wi-Fi, GSM can provide accurate location information as well. Lastly, GPS signals are used to identify outdoor locations where Wi-Fi and GSM signals are not present or unique. Since each of these technologies have different energy requirements, they are used in a staged fashion that allows a user to run the location recognition continuously without draining the phone's battery.

3.3.1.2 Trip Recognition

The location of a user is an important piece of information for both users and service providers. Similarly, having information about the mode of locomotion between two locations can be beneficial for service providers. Knowing how trip was done – i.e. whether the user went on foot, took a car or a bus, stood in the bus or was able to find a seat – can help public transit providers to offer better services. In order to determine the mode of locomotion, the GAMBAS middleware encompasses multistage classifiers which integrate different sensors including accelerometers, Wi-Fi scans and GSM cell-IDs. Thereby, the classifiers use accelerometer samples to identify the general motion of the user. This allows them to determine if the user is walking, running, climbing stairs, etc. If a continuous detection of walking or running is detected between the locations, they can derive that the user was traveling on foot. If the user is not walking, the trip recognition components are using Wi-Fi and GSM cell information to estimate the movement speed of the user, which can then be used to narrow down the remaining modes (e.g. driving in a car, riding a bus, etc.).

Given a suitable infrastructure, such as the one deployed in the city of Madrid, it is even possible to identify the actual vehicle type (e.g. a specific public bus running on a particular bus line). However, even if this infrastructure is not available, it is still possible to derive the movement modality with high accuracy. In order to measure the accuracy of configuration for the trip recognition, we performed a number of validation tests over the data gathered from different modes of transportation. The final classifier with the overall of accuracy of 91.4% and the confusion matrix are shown in Figure 3.13 and Figure 3.14, respectively.

These results have been gathered by capturing training data from 4 persons in Duisburg and Bonn over the course of multiple days. Consequently, there might be a bias regarding the fit for this particular area and overall the results may be worse when applied to different areas or users.

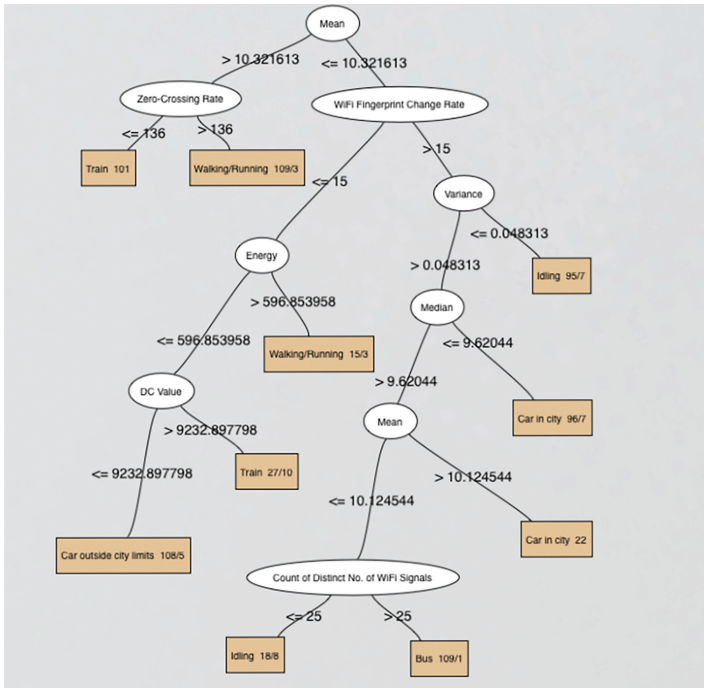

Figure 3.13 Trip Recognition Classifier.

	Bus	Walking/Running	Train	Idling	Car in city	Car outside city
Bus	108	0	3	9	3	4
Walking/Running	1	115	1	1	0	0
Train	0	0	116	4	0	3
Idling	3	0	0	98	4	0
Car in city	2	1	2	3	106	2
Car outside city limits	1	3	5	1	6	95

Figure 3.14 Trip Recognition Confusion Matrix.

However, given the high accuracy of the results, it is conceivable that this approach is broadly applicable in general.

3.3.1.3 Sound Recognition

The sound recognition components make use of audio-data collected on the mobile device and combine it with location data. They can be used for two major purposes. First, they can be used to identify features of the user's environment as done with noise recognition. Second, they can be used in

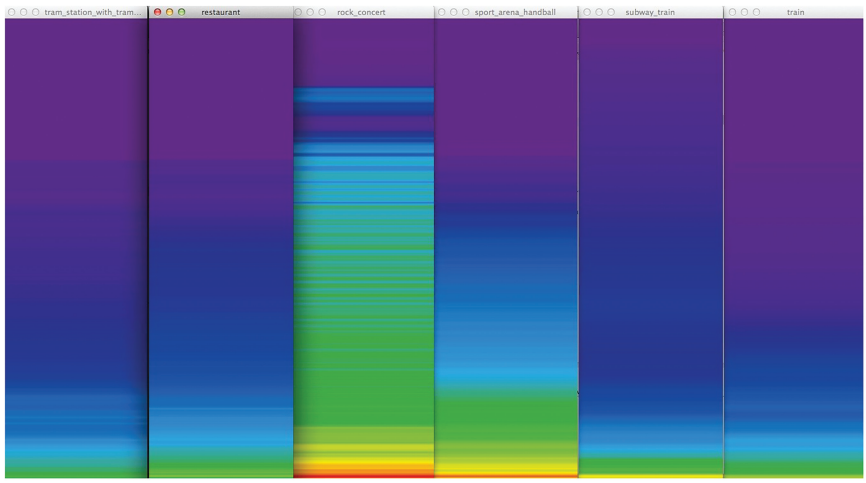

Figure 3.15 Average Frequency Vectors (Train Station, Restaurant, Rock Concert, Sport Arena, Subway, Train).

order to provide a natural way of performing user inputs as done with Voice Tagging or Voice Control.

3.3.1.3.1 *Noise Recognition*

There are several user contexts that come along with a characteristic sound environment. Being on a crowded bus, for example, a person is surrounded by a constant bus engine sound as well as human voices and other noises created by a crowd of people. This can be exploited to extract information about the user context from audio collected on the mobile device as well as to gather information about the public transport traffic situation in the whole city. To do this, we collected data sets using mobile devices carried around the city by test users. The devices are used to record several distinct audio environments like crowded bus stations and traffic jams. The collected data is then used to create sound profiles of different environments, e.g. crowded, not crowded, rush hour, etc. To do this, we compute an average frequency vector from individual samples. As shown in Figure 3.15, the average frequency vectors are different depending on the characteristic sounds present in an environment.

In order to classify recordings, the noise recognition components compute the average frequency vectors of new samples and compare them with the known profiles using the Euclidean distance between the new and all known

vectors. In order to minimize the number of comparisons, we use K-Means clustering to reduce the candidates to one (good) representative for each sound profile.

3.3.1.3.2 *Voice Tagging*

Every person typically has certain locations that he attends frequently, for example, his home or his work place. To enable the user to enter these locations as destinations in a more efficient way, the voice-tagging component enables users to speed up repeated inputs. In a first step, it allows the user to add a short audio tag to his current whereabouts. For this purpose, an application typically offers a button saying "voice tagging", which, when pressed, starts a short audio recording. Typically the audio input will contain a sequence of one to three words spoken by the user. This audio is then stored in the database in a reduced form, together with the current geo data, provided by the location recognition component. At any later point in time, the user can refer to his audio tag by speaking the words used for the tag again. So if he had tagged a place by saying "my favorite restaurant", he would just have to phrase these words again to select the tagged location. At first sight, this component looks like a speech-recognition-application. However, the required computations to perform the matching between the stored voice tags and the user input are much simpler. In addition, the integration of voice tags into an application is also easier, since it does not require the definition of a grammar that defines the possible inputs. However, in contrast to voice control, voice tagging requires more effort on the side of the user, since the user has to set up tags in advance to be able to use his or her voice as an input.

3.3.1.3.3 *Voice Control*

The idea of the voice-control component is to enable the user to tell the application where he wants to go next by simple speech input. Typically, the component is activated via a voice-control button in the user interface. Once the button is pressed, it will start to receive audio data and return the recognized location. A typical speech input would be "I want to go to the main station." To enable voice control, we have integrated a customized version of the Sphinx speech recognition engine for which we developed custom models to support different target languages (including German, Spanish and English). In addition, we have developed a custom grammar for the applications described in Chapter 6. Due to the specific application scenario, the grammar includes a general list of public transport stations in the city

of Madrid and it contains template sentences that are frequently used to specify routing targets such as “how do I get to Moncloa” or “compute a route to Atocha”.

3.3.2 Intent Recognition

The intent recognition components take the recognized locations and trips and provide future predictions on them. Knowing the current location and mode of user transport provides significant opportunities to the service providers to improve their business, but the added ability to predict how long the user will stay at a particular place and what would be his next destination could help service providers to offer even more useful services. Apart from the service providers, a user can have many personal applications that can take advantage of this information. For instance, there can be a device charge reminder application which can alert the user to charge the batteries, based on the predicted duration of his stay at the current location and also his intended next destination. With respect to intent recognition, the acquisition framework provides duration prediction and destination prediction components.

3.3.2.1 Duration Prediction

Knowing how a long a user will stay in a particular place requires storing user location and running an offline analysis to compute predictions for the duration of user’s stay in the same place in the future. There can be different options to store information about user’s stay in a particular location, e.g. this information can be stored in users’ device, in the cloud and also at third-party trusted servers. Clearly, storing such information elsewhere than on the user’s device is prone to privacy issues and thus for the scope of GAMBAS, this information is only stored on the user’s device. During the training phase, whenever a user visits a new place or a place that he has visited before, the duration prediction component stores how long the user stayed there, at which day of the week and at, which time of the day. The system then performs offline analysis on this data in addition to previously stored data which includes the information about the frequency of the user’s visit to that location and his usual next locations. The system then runs a prediction algorithm to compute new predictions or update existing ones. In order to minimize the impact on the battery of the user’s device, the offline analysis of data is usually done whenever the device is plugged to a socket and charged for a longer time, e.g. during the night.

3.3.2.2 Destination Prediction

In addition to knowing the current location and stay of a user in a particular location, the ability to predict the next user destination is also very useful. This information can be used to compute the transport routes proactively, for example. Similar to the duration prediction, the destination prediction is performed by analyzing the history of places visited by the user. This mainly involves identifying some sequence in the places visited by the user, the time of the day, the day of the week, frequency of visits, etc. For example, we can predict that every Saturday the user first goes shopping, then goes to a fitness club and afterwards meets friends and family. Similar to the duration prediction, this information is stored on the device and offline analysis (when the device is being charged) is performed to compute new predictions or update the existing ones.

3.3.2.3 Prediction Algorithm

The prediction algorithm uses three prediction techniques, namely time series prediction, least k history predictor and a location-dependent Markov model. The time series prediction works by taking into account the history of visits by a user to a particular location. Each visit to a particular location is saved and marked by the starting time and the duration of stay at that location. In order to predict the starting time when the user is likely to visit that location again, we choose latest last m values of starting times from the history of visits. We than identify subsets of m values of starting times in the history of visits and identify sets that are close to the latest last m values. The predicted value for next user visit to that location is obtained by averaging the next starting time value following the sets of m values. At the end of this exercise, we have a set of predicted starting time of all the locations that the user might visit. In order to select a unique next location, we check whether the predicted starting time of a location is under some time threshold T. If we can find such a starting time, we select the associated location to be the next possible location. If more than one predicted locations satisfy the criteria, we choose one randomly. A similar approach is also used for determining the duration of stay. In our tests with multiple users, the prediction techniques typically range around 20–40% accuracy, depending on the regularity of the movement patterns of the user.