The NARF Architecture for Generic Personal Context Recognition

Marcus Handte, Umer Iqbal, Wolfgang Apolinarski, Stephan Wagner and Pedro José Marrón
Networked Embedded Systems Group
University of Duisburg-Essen
Duisburg, Germany
Email: {firstname.lastname}@uni-due.de

***Abstract*—Ubiquitous computing envisions applications that provide seamless and distraction-free support for everyday tasks. To achieve this goal, applications must be able to adapt to their environment and to the intents of their users. Thereby, they need to automate adaptation decisions to minimize the resulting distraction. As a consequence, it is necessary to acquire an understanding of the user's current situation to ensure that the automation results in a desirable application behavior. The wide-spread use of personal mobile devices provides a promising technical basis to acquire this knowledge in a seamless manner. However, in order to account for the resource limitations of such devices, existing personal context recognition systems are typically highly specialized and cannot be adapted easily to different scenarios. In this paper, we motivate the need for an generic personal context recognition system and we derive the requirements on such systems. As an approach towards creating such a system, we describe NARF - our ongoing effort to create an adaptive context recognition framework. To assess the benefits and limitations of the architecture, we discuss an application that we have built using our current prototypical implementation of this framework.**

I. Introduction

Ubiquitous computing envisions applications that provide seamless and distraction-free support for the everyday tasks of their users. To achieve this goal, applications must be able to provide task support in vastly different environments and they must be able to consider different sets of user intents. Intuitively, both – the environment and the intents – are often not static. Instead, they are changing at runtime depending on the user's situation. To provide a truly seamless experience, ubiquitous computing applications must be able to adapt to such changes at runtime and this adaptation must be done automatically in order to achieve the goal of being distraction-free.

To ensure that automatic adaptation results in a desired behavior, applications need to be able to perceive the relevant parts of the user's situation. From a technical perspective, this requires applications to determine the set of available devices which can then be used to determine viable modes of execution. However, in cases where multiple modes are possible, selecting the optimal one requires an understanding of the non-technical features of the user's situation. This set of features is commonly referred to as the context. Examples can be the current location, activity and future plans of a user.

The ubiquity of personal mobile devices provides a promising technical basis to determine the context of users in an automated manner on a large scale. The reason for this is threefold. First and foremost, a personal mobile device is directly associated with a particular user. Since the user carries the device continuously during the day, many features of the device context can be used as a representation of the user context without further ado. Secondly, more and more devices are equipped with sensors. Examples of such sensors are gyroscopes, accelerometers, cameras and microphones, to name a few. At the same time, an increasing number of devices is able to use wireless communication technology such as GPRS or UMTS to access relevant on-line information sources like calendars, task lists or maps at any point in time. Together, this creates an unprecedented richness of physical and virtual information sources that can be tapped to determine the context of a user *seamlessly*. Last but not least, although personal mobile devices are often considered resource-poor when compared to traditional computers, the available resources are usually underutilized in normal operation.

In the recent past, these factors have contributed to the development of a number of context recognition systems for personal mobile devices. Typically, these systems combine the information provided by different physical sources to recognize a particular feature of the user's context. However, in order to support resource-poor personal mobile devices, the existing recognition systems are manually fine-tuned to a narrow scenario and they cannot be adapted easily to changing requirements. This, in turn, introduces a limitation on the practical applicability of the systems since the features of the context that are relevant and the way how they can be recognized efficiently may be highly user- and scenario-specific and they often change dynamically at runtime. As a simple example consider that the activities performed by different users depend partially on their job and even with respect to a single user, the set of activities that may be performed will usually change depending on the time of day.

To improve the practical applicability of personal context recognition, the recognition itself must be generic. In this paper, we derive the resulting high-level requirements on personal context recognition systems and as a possible incarnation, we discuss the architecture of NARF - our ongoing effort to build an adaptive context recognition framework. To assess

the architecture, we discuss the benefits and limitations using an application that we have built using our current prototype.

The remainder of the paper is structured as follows. In the next section, we briefly review the state of the art in context recognition with mobile devices. Thereby, we motivate the need for generic personal context recognition. In Section III, we derive the resulting requirements. In Section IV, we describe the architecture of NARF, our adaptive context recognition system. In Section V, we assess the benefits and limitations of the framework using an application that we built with our current prototype. Finally, in Section VI, we conclude the paper with brief summary and an outlook.

II. Related Work

In the recent past, researchers have developed a number of context recognition systems on the basis of personal mobile devices. Some examples are [4], [5], [11], and [13]. Usually, these systems fix the features that ought to be detected at design time. The resulting systems are then manually tuned to achieve an optimal recognition result in a particular scenario. As a consequence, they cannot be adapted easily to different scenarios without additional manual engineering which limits their applicability.

Many existing systems such as [11], [12], [15], [10], [9], [17], [3] use built-in sensors in mobile phones to recognize the desired context features. For instance, CenceME [11] uses built in microphone and accelerometer to detect different user states. These user states are then injected into on-line social forums. Nericell [12] uses accelerometers, microphones, and GPS to detect uneven roads and traffic conditions. Vtrack [15] is a traffic congestion monitoring systems which works on a collective participation of road users. Users with their mobile phones capture traffic situation and send it to a centralized server. The server sends up to date information to other cars. SoundSense [10] uses microphone for recognizing different sounds and employs supervised and unsupervised learning technique for identifying different sound types. These and other systems like [9], [17] and [3] provide satisfactory performance in their respective scenarios but are unable to adapt to new requirements.

A basic limiting factor of generic context recognition systems for personal mobile devices is the limited availability of resources. Recently different approaches to minimize the resource consumption during context recognition have been developed. For example, [7] uses a bidirectional approach for context monitoring. The main idea behind the approach is to detect changes in the context 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. [16] suggests using hierarchical sensor management strategy to detect user states and state transitions. To do this, the system associates a set of sensors with a particular state and it activates them only if the state is detected. [2] presents a framework in which sensor state detection is structured as decision tree classifier. The sensor sampling rates are adjusted dynamically so that only necessary data is collected at any given time. The energy efficiency is achieved by structuring classification as a series of queries for different stages of the classifier, each requiring a different number of decisions.

Beyond these rather specialized tools, there are also a few rapid prototyping tools for context recognition systems that are geared towards generic support. As such, these systems aid developers in creating context recognition systems according to different requirements. For instance, [14] provides a uniform abstraction for applications to access and use context information. Thereby, the tool kit hides the actual context sensing and its interpretations from the applications. Similarly, [1] provides functionalities to develop distributed context recognition systems. Thereby, it simplifies the development process with its reusable components and a set of parametrized algorithms, filters, and classifiers. However, while [14] is primarily targeted at context recognition with predeployed sensors, [1] is targeted towards motion recognition with specialized wearable systems. In contrast to this, [8] is a data gathering and processing open source platform which is specifically targeted towards mobile phones with varying hardware capabilities. Extensibility is achieved by means of a minimal core that is extended via plug-ins. The system relies on a black-board architecture that can be configured during start up phase, however, at runtime the system cannot be adapted.

As discussed above, there exists a number of context recognition systems that has been tailored to the specifics of mobile devices. These systems clearly indicate the viability of personal context recognition in specific scenarios. Furthermore, there are a number of approaches that show possible ways of reducing the resource consumption which improves the viability. In addition there are also a few tool kits and systems that strive for generic context recognition. Although these systems are either not tailored towards personal mobile devices or they only provide limited flexibility, they clearly indicate the usefulness of genericity.

III. Requirements

In the following, we discuss the requirements on personal context recognition. These requirements can be derived directly from the technical characteristics of personal mobile devices and the desire of supporting context recognition in a generic manner.

- *Uniformity*: As indicated in the introduction, many ubiquitous computing applications benefit from or require personal context recognition. However, the features that are relevant for these applications vary depending on their purpose. For example, a navigation application might be interested in the speed of the user to switch between visual and audio mode whereas a calendar application might be interested in the surroundings of the user to determine whether and how an upcoming appointment should be signalled. Thus, in order to be useful for a broad range of applications, personal context recognition

should support the recognition of different features in a uniform manner.

- *Extensibility*: Since there are numerous features that may be needed by different applications, it is unrealistic to assume that a single developer is able to provide all required recognition methods. As a consequence, a personal context recognition system should be extensible and it should enable the joint use of recognition methods that have been developed in isolation. Furthermore, to ease the development of new recognition methods, the system should enable the reuse of parts of the existing methods when appropriate.
- *Configurability*: As discussed previously, existing recognition methods often propose a particular set of parameters to balance the trade-off between accuracy and resource utilization, for example. Yet, this optimization is usually scenario specific. In practice, different parametrizations may be optimal for different scenarios. Moreover, depending on the scenario there may be several alternative methods for recognizing the same context feature. As a simple example consider that location can be detected using different technologies such as GPS in outdoor scenarios or Wifi in indoor scenarios. Consequently, a system for personal context recognition should be highly configurable to be applicable in a broad spectrum of scenarios.
- *Efficiency*: Due to the use of personal mobile devices as technical basis, the resource usage for context recognition must not hinder the primary function of the devices. As a consequence, personal context recognition must make efficient use of resources. Towards this end, the recognition system should not spend resources for recognizing irrelevant features. In addition to that, it should perform the recognition of the relevant features with minimal effort.

IV. Architecture

In the following, we introduce the architecture of NARF, our ongoing effort to build an adaptive context recognition framework. The goal of NARF is to provide a generic platform for personal context recognition. Towards this end, the framework aims to satisfy the requirements discussed in the previous section. An overview of the architecture is shown in Figure 1. As shown in this figure, the framework consists of two main building blocks, namely a runtime system which is deployed on a personal mobile device and an associated set of development tools that is used off-line to create the necessary configuration.

The task of the runtime system is twofold. During normal operation, the runtime system performs the actual recognition of the relevant features of the context and it enables applications to retrieve them and to be notified upon changes. In addition to the normal operation that performs the actual recognition, the runtime system can also be used to gather traces in sampling mode. These traces can then be used as an input for the development tools.

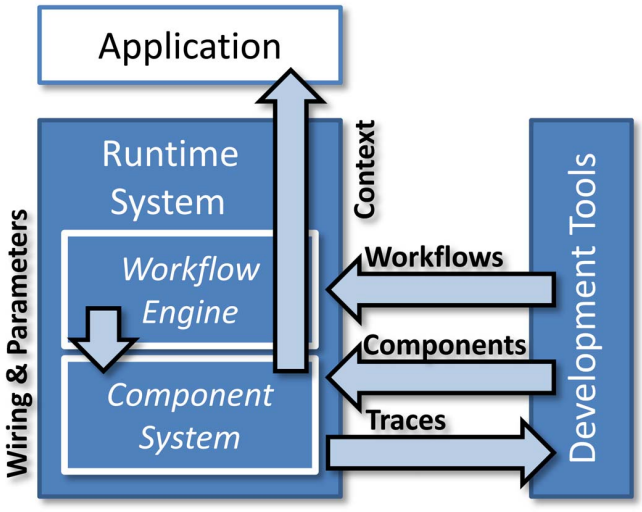

Fig. 1. NARF Architecture

To perform the recognition of context, the runtime system must be configured. Performing this configuration is the main task of the development tools. In addition to the configuration of the runtime system, the development tools are also used to enable the development of new recognition methods. To simplify this, the development tools facilitate the reuse of recognition logic that has been developed previously.

In the following, we describe both, the runtime system and the development tools in greater detail and we discuss how they interact in order to realize the requirements identified in Section III.

A. Runtime System

As hinted in the previous section, the main task of the runtime system is to perform the context recognition. To do this, the framework introduces two different systems that are stacked on top of each other. As shown in Figure 2, a component system forms the lower level of the runtime system. On top of the component system, NARF introduces a work flow abstraction that is realized by a minimal work flow engine.

The main task of the component system is to perform the recognition of a set of features in an efficient manner. To do this for arbitrary features, the component system abstracts form the specific recognition method by introducing three different abstractions, namely components, parametrizations, and wirings. In the following, we briefly describe their structure and purpose:

- *Components*: In NARF, components are the (reusable) building blocks of the recognition methods. Conceptually, a component consists of an implementation that exposes typed input and output ports that may be wired to the ports of other components. Thus, the ports enable components to interact with each other in a controlled manner. To do this, a component may post a value to an output port or it may read a value from an input port.

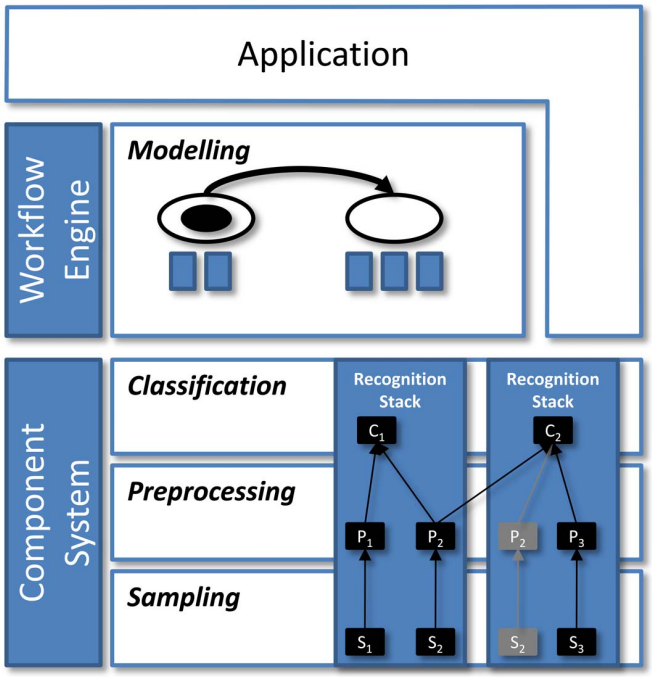

Fig. 2. NARF Runtime System

- *Wiring*: In order to be reusable, components are isolated from each other by means of ports. However, the recognition of a feature often requires the combination of multiple components in a specific way. Wirings express such combinations by determining how the input and output ports of a set of components shall be wired.
- *Parametrization*: In addition to ports, a component may expose parameters that can be used to adapt its internal behavior. However, the parameters are not exposed to other components but, they can be accessed and manipulated by the component system itself. Developers may define different parametrizations for a particular usage of a component.

In general, it is possible to categorize the components according to their function into three different levels. As shown in Figure 2, at the sampling level a component may provide access to a physical or virtual data source that can be used to sample data. Components at the preprocessing level perform generic tasks such as noise-reduction or feature-extraction on the raw data provided by the sampling level. On top of the preprocessing level, classification components combine multiple features to recognize the target feature of the context.

Except for the most basic methods, most feature recognition methods have to combine a number of components at all of the levels. Thus, it is possible to represent the basic configuration for a recognition method by means of a wiring. For example, recognizing a noisy environment will require a sampling component to capture audio from a microphone, a preprocessor component to extract the power level and a classifier component to decide whether the environment is rather noisy or calm.

However, most of the components usually exhibit parameters that can be used to control their behavior. Thus, in order to describe the complete configuration for a feature recognition method, it is necessary to provide the parametrization for each component of the wiring. For the previous example, such a parametrization might define the sampling rate and frame size for the sampling component or the threshold for the power level used by classifier component.

In order to recognize multiple features simultaneously, the component system is able to execute multiple wirings with their associated parametrizations. In cases where the recognition methods share similar parts of the configuration, the component system tries to merge the wirings intelligently to avoid duplicate computation. This ensures that the recognition is performed efficiently even in cases where the recognition methods have been developed independently. An example for this is shown in Figure 2. Since the depicted recognition configurations share the sampling component S2 and the preprocessor P2, the component system only executes them once. In addition, it changes the wirings to use the output of the same preprocessor P2 in both configurations.

Intuitively, the automated merging requires a runtime analysis of the configurations to detect overlapping graphs. In addition, it is necessary to determine whether there are differences in the parametrization. Our current implementation only supports automated merging of identical parametrizations, however, in many cases it is possible to introduce transformation components into the graphs to merge different parametrizations as well. For example, in order to bridge between different sampling rates, a transformation component could skip some of the samples.

Using the outlined component model and the associated merging mechanism, the component system is able to support the development and execution of arbitrary recognition methods in an efficient manner. Thereby, the component system executes the configured recognition continuously. However, depending on the state of the environment, an application may only be interested in a subset of the features at a time. In such cases, the continuous execution of all recognition methods would be inefficient.

To avoid this, the framework enables applications to specify the desired recognition methods in a context dependent manner. Instead of specifying a static set of recognition methods, an application specifies a work flow consisting of states to model requirements on the recognition and transitions to model the possible ways of switching between the states. Although this may seem complicated at first, it enables the application developer to apply fine-grained control over the recognition. If this fine-grained control is not needed, it is possible to specify a static set of requirements by introducing a single state without transitions. In the following, we briefly describe the purpose and structure of the flows used by NARF.

- *State*: Work flows in NARF consist of a set of states. Each state is associated a particular requirement regarding the recognition methods that shall be executed by the component system. To do this, each state may be associated with

a set of wirings and parametrizations for the component system. One of the states is marked as initial state and this state is active when the flow is added. Conceptually, the states model different situations that require the detection of a distinct set of context features.

- *Transition*: In addition to states, the work flows may exhibit transitions between states. The transition can be associated with conditions over the context features that are currently detected. Intuitively, this reduces the set of features to the ones specified in the source states of the transition. A transition is taken if the source states are active and the specified context features are detected.

The runtime system introduces a simple work flow engine to execute the flows. The main task of the work flow engine is to compute the active states of the flow and to trigger state changes when a transition is fired. To trigger these changes, the work flow engine relies on the component system to detect the context features specified in transitions. Once a state becomes active or inactive due to a transition, the work flow engine automatically stops the wirings and parametrizations defined by the inactive state and it starts the wirings and parametrizations of the active state.

In order to start the recognition, an application may inject a work flow in the engine at runtime. Once the work flow is executed, an application may receive changes to the features that are detected by recognition methods that are executed by the component system. In addition to that, an application may also monitor the state transitions in the work flow engine. Thus, an application can use the states in the work flow to summarize a particular set of changes to context features.

B. Development Tools

To fully utilize the flexibility of the runtime system, the framework encompasses a set of associated development tools. Due to the modularity of the runtime system, there are three tools that target different aspects. These tools are depicted in Figure 3 and they target component development, wiring and parametrization as well as work flow specification.

A component interface generator simplifies the development of components by generating the necessary stub code using a graphical component wizard. This interface code can then be implemented using a standard editor. To generate the stub code, the developer defines its interface by selecting the type of component, i.e. sampling, preprocessor, classifier, the type and number of ports as well as the component parameters.

Beyond the component generation, a wiring and parametrization tool is used to derive a wiring with an appropriate parametrization as well as a classifier. To do this, the tool requires tagged traces. The untagged traces can be generated by capturing the raw samples of sampling components deployed in the runtime system. In order to capture traces, the runtime system supports a tracing mode that stores the samples on local storage. After retrieval from the device, the traces need to be manually tagged with the features that shall be detected. Once tagged, the wiring and parametrization tool uses the trace to determine a suitable wiring and parametrization.

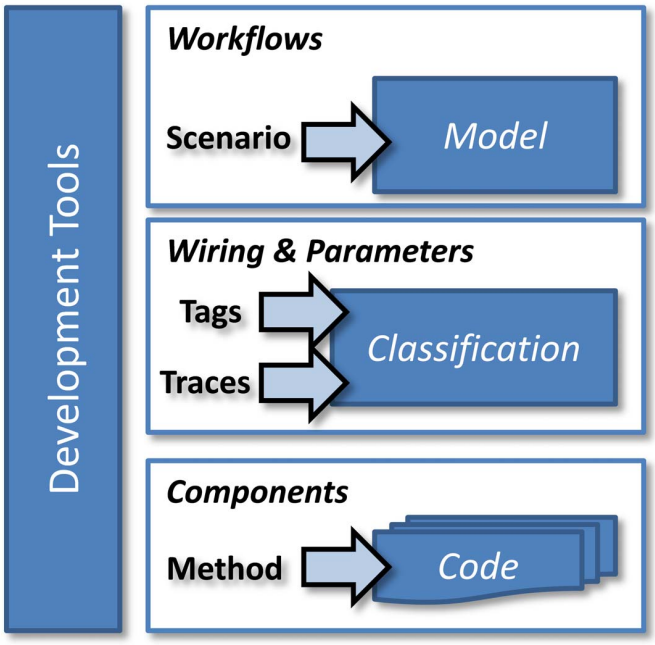

Fig. 3. NARF Development Tools

To automatically compute this, the tool runs different wirings of preprocessors with different parametrizations against the trace. Thereby, it tries to create a classifier that results in the associated tags. Without manual help from the developer, this requires the testing of all possible preprocessors with all possible parametrizations. Thus, depending on the number of preprocessors and the size of their parameter spaces, this can be a computationally intensive task. Minimizing the resulting search space is one of our future research efforts. A key idea to do this is to leverage monotonic behavior in the parameter domains.

Once the wiring and parametrization has been generated either manually or through the help of the development tools, it can be used within a work flow specification. To simplify the specification of work flows, a graphical tool can be used to specify the states and the transitions. Thereby, each state must be associated with a particular set of wirings with parametrizations that shall be executed. Similarly, the transitions must be configured with a condition that must hold true for it to be triggered. The resulting work flow can then be injected into the work flow engine to start the corresponding detections.

V. Assessment

In order to assess the architecture, we have started to implement the individual parts and we have built a simple example application. Our current implementation of NARF has been written in Java and it is targeted towards the Android platform. At the present time, the implementation encompasses a simplified version of the runtime system which we are currently refining by implementing additional recognition

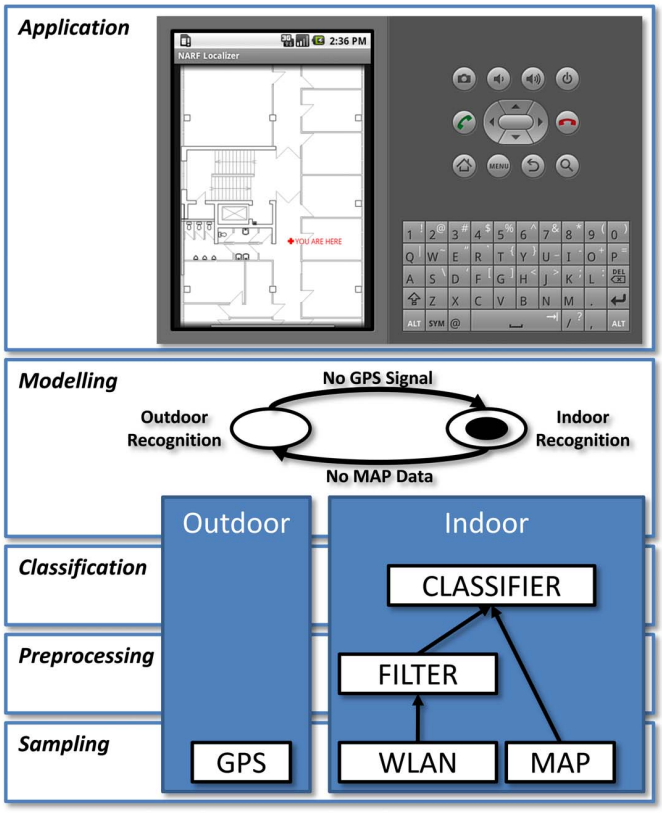

Fig. 4. Example Application

methods for different context features. The next step is to implement the front ends for the development tools which should drastically simplify the use of the runtime system.

In the following, we first describe a simple example application that we have built using our current version of the runtime system. Thereby, we show how the component system and the work flow engine can be put to use in practice. Thereafter, we revisit the requirements set out in Section III and we discuss why and how they are fulfilled by the architecture.

A. Example

To validate the architecture of the runtime system, we have implemented a simple example application. The example application recognizes the location of the user using a personal device that is running NARF. It then visualizes the user's location in a moving map. Although, the application itself is not complicated, it uses many features of the framework.

Intuitively, there may be multiple alternative ways of retrieving the location information. In outdoor scenarios, the localization by means of GPS is a natural choice due to its wide availability and simplicity. However, in indoor scenarios GPS cannot be used. Thus, in order to provide location information in a seamless manner, it is necessary to use an alternative recognition method. Due to its wide availability, a rather common approach is to use 802.11 technology for indoor localization and there are several possible ways of doing this. For the example, we decided to use fingerprinting due to its simplicity. The bottom of Figure 4 shows the stacks resulting from the usage of these recognition methods.

The stack for the outdoor localization, depicted on the left side of Figure 4, solely consists of a single component at the sampling layer. The reason for this is that the GPS hardware is already performing the necessary preprocessing and classification internally. Thus, the GPS component can simply retrieve the current coordinates from the GPS receiver which can then be made available to the application.

The stack for the indoor localization, depicted on the right side of Figure 4, is more complicated. At the sampling layer, it consists of a WLAN component and a MAP component. The WLAN component is responsible for generating fingerprints by scanning the available 802.11 networks to determine their signal strengths. In order to derive the current user location from the fingerprint, the MAP component provides access to a list of fingerprints with associated locations. To do this, the MAP component uses BASE [6], our middleware for spontaneously networked devices which enables the component to retrieve the list from a local server. At the preprocessing layer, a filter component is responsible for improving the fingerprint by removing irrelevant access points. The resulting fingerprint is then used by the CLASSIFIER to determine the location using the list provided by the MAP.

Since a user can either be located indoors or outdoors at any point in time, running both recognition methods simultaneously results in unnecessary resource usage. To minimize the resource usage, the application specifies its requirements with a work flow that models the fact that the recognition methods should not be executed simultaneously. This can be done by introducing a flow with one state for each method and appropriate transitions.

Intuitively, the transitions need to be refined with conditions that define when to switch between the states. As explained in Section 2, a key limitation is thereby that the conditions may only refer to context features that are detected in the source state of the transition. Otherwise, the work flow engine would not be able to detect the feature by means of the component system. Thus, for this example, we define the conditions in such a way that they are triggered when the localization is no longer successful, i.e. if the GPS component does no longer receive coordinates or if the map component does no longer provide a list of fingerprints. Clearly, this may result in oscillations in cases where both, the indoor and the outdoor location, cannot be performed. However, in such cases the interleaved scanning is a desirable behavior.

In order to use the recognized features, the application may access the location provided by the component system as well as the current state of the work flow engine. Our example application uses both. The state of the work flow engine is used to adapt the way how the application tries to retrieve a map. If the outdoor recognition is used, the application can simply retrieve a map through a web service such as Google maps. If the indoor recognition is used, the application uses BASE in order to download an indoor map from a server that is deployed locally in the current environment. The location

provided by the component system is then used to mark the current user position in the map.

B. Discussion

In the following, we discuss how the architecture addresses the requirements identified in Section III. To clarify the individual points, we revisit the example application described previously.

- *Uniformity*: As discussed previously, the NARF architecture is independent from the context features that shall be recognized. It enables uniform access to the recognized features by means of supporting arbitrary access to all types of data that are generated within a recognition stack. The example application solely uses the information that is generated at the top most component of each stack. However, other applications might benefit from information that is generated by the intermediate components. As an example consider that it might be useful to visualize the list of available fingerprints to indicate the granularity of the localization.
  In addition, the framework enables applications to adapt the recognition using a work flow model. Although the primary use case for this restriction is to minimize the resource usage, exposing the current state of the work flow engine can provide valuable information. In the example application, the state of the work flow engine is used to switch between map providers. In general, it can be used for grouping the access to a set of context features.
- *Extensibility*: The NARF architecture supports extensibility at several levels. At the component level, additional components can be implemented to provide access to physical data sources, such as a 802.11 interface, virtual data sources, such as a local map server, preprocessing algorithms and classifiers. Above the component level, developers can extend the framework by providing different wirings and parametrizations to recognize a context feature. Thereby, it is possible to develop the recognition methods independently since the runtime system can execute multiple methods simultaneously.
- *Configurability*: In order to adapt the recognition to a particular scenario, the NARF architecture introduces work flows. The work flows describe which recognition methods should be used at a time. By defining different parametrizations, it is also possible to use the same recognition method with different trade-offs in a single application. To simplify the development of alternative parametrizations, the runtime system can be extended with additional development tools. Thereby, it is possible to use the lower levels of the runtime system to simplify the generation of traces.
- *Efficiency*: To support resource efficient recognition, the runtime system of NARF provides several manual tuning knobs and automatic mechanisms. Using work flows, it is possible to manually define the set of recognition methods that is relevant in a particular context. By determining the current state of the flow, the runtime system can then automatically start and stop the recognition methods. Similarly, by enabling developers to provide different wirings and parametrizations, they can define different configurations to balance the trade-off between accuracy and resource usage. These can then be used as part of the work flow definition.
  Besides from reducing the resource utilization by optimizing the set of recognized features, the component system also minimizes the resource overhead resulting from the isolated development of recognition methods. To do this, the component system automatically merges the recognition graphs of several features, in cases where they share the same components. This enables the system to execute arbitrary combinations of recognition graphs in an efficient manner.

VI. Conclusion

To provide seamless and distraction-free support to their users, ubiquitous computing applications need to adapt to dynamic environments and changing user intents. To do this, applications need to be able to determine the technical as well as the non-technical features of their context. Personal mobile devices are a promising technical basis to acquire the non-technical features in an automated manner. However, doing this in a large scale requires novel personal context recognition systems that can be adapted to different user requirements and scenarios.

The architecture of NARF is specifically geared towards generic yet efficient support for context recognition. Thereby, the framework combines off-line configuration by means of development tools with runtime adaptation by combining a component system with a work flow engine. By using work flows to model relevant sets of features, the component system can minimize the resource consumption of context recognition.

At the present time, we are refining the prototypical implementation of the runtime system and we are implementing the development tools. Simultaneously, we are developing a set of standard components that minimize the manual implementation needed for the development of further recognition methods.

Acknowledgment

This work has been partially supported by CONET (Cooperating Objects Network of Excellence) funded by the European Commission under FP7 with contract number FP7-2007-2-224053.

References

[1] David Bannach, Oliver Amft, and Paul Lukowicz. Rapid prototyping of activity recognition applications. *IEEE Pervasive Computing*, 7:22–31, 2008.

[2] Ari Y. Benbasat and Joseph A. Paradiso. A framework for the automated generation of power-efficient classifiers for embedded sensor nodes. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 219–232, New York, NY, USA, 2007. ACM.

[3] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G-S. Ahn, and A. T. Campbell. The bikenet mobile sensing system for cyclist experience mapping. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 87–101, New York, NY, USA, 2007. ACM.

[4] Jon Froehlich, Mike Y. Chen, Sunny Consolvo, Beverly Harrison, and James A. Landay. Myexperience: a system for in situ tracing and capturing of user feedback on mobile phones. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 57–70, New York, NY, USA, 2007. ACM.

[5] Hans W. Gellersen, Albercht Schmidt, and Michael Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. *Mob. Netw. Appl.*, 7(5):341–351, 2002.

[6] Marcus Handte, Christian Becker, and Gregor Schiele. Experiences - extensibility and minimalism in BASE. In *Proceedings of the Workshop on System Support for Ubiquitous Computing (UbiSys) at Ubicomp*, 2003.

[7] Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Hyonik Lee, Youngki Lee, Souneil Park, Taiwoo Park, and Junehwa Song. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 267–280, New York, NY, USA, 2008. ACM.

[8] Joonas Kukkonen, Eemil Lagerspetz, Petteri Nurmi, and Mikael Andersson. Betelgeuse: A platform for gathering and processing situational data. *IEEE Pervasive Computing*, 8(2):49–56, April-June 2009.

[9] Michael Lawo, Otthein Herzog, Paul Lukowicz, and Hendrik Witt. Using wearable computing solutions in real-world applications. In *CHI '08: CHI '08 extended abstracts on Human factors in computing systems*, pages 3687–3692, New York, NY, USA, 2008. ACM.

[10] Hong Lu, Wei Pan, Nicholas D. Lane, Tanzeem Choudhury, and Andrew T. Campbell. Soundsense: Scalable sound sensing for people-centric applications on mobile phones. In *MobiSys '09: Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, pages 165–178, New York, NY, USA, 2009. ACM.

[11] Emiliano Miluzzo, Nicholas D. Lane, Shane B. Eisenman, and Andrew T. Campbell. Cenceme - injecting sensing presence into social networking applications. In *EuroSSC*, pages 1–28, 2007.

[12] Prashanth Mohan, Venkata N. Padmanabhan, and Ramachandran Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336, New York, NY, USA, 2008. ACM.

[13] Petteri Nurmi, Joonas Kukkonen, Eemil Lagerspetz, Jukka Suomela, and Patrik Floréen. Betelgeuse: a tool for bluetooth data gathering. In *BodyNets '07: Proceedings of the ICST 2nd international conference on Body area networks*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[14] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM.

[15] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98, New York, NY, USA, 2009. ACM.

[16] Yi Wang, Jialiu Lin, Murali Annavaram, Quinn A. Jacobson, Jason Hong, Bhaskar Krishnamachari, and Norman Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 179–192, New York, NY, USA, 2009. ACM.

[17] Jamie Ward, Paul Lukowicz, Gerhard Troster, and Thad Starner. Activity recognition of assembly tasks using body-worn microphones and accelerometers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1553–1567, 2006.