A Generic Approach for Energy Efficient Context Recognition using Smart Phones

Muhammad Umer Iqbal, Marcus Handte, Pedro José Marrón
Networked Embedded Systems, University of Duisburg-Essen, Germany
umer.iqbal@uni-due.de, marcus.handte@uni-due.de, pjmarron@uni-due.de

***Abstract*—Intelligent environments rely on context information for providing services to their users. Among various existing platforms for context recognition, smart phones are one of the most widely used. However, despite numerous advantages, smart phones exhibit limited energy resources. To mitigate this there exist approaches for energy efficient context recognition for smart phones but these approaches are usually fine-tuned for specific types of context. As a result their applicability to other context types is limited. In this paper, we present an energy efficient context recognition system for smart phones which provides generalized applicability of four generic energy efficiency techniques which have been described in the literature and applied to location sensing using mobile phones. These four techniques as termed by authors include Suppression, Substitution, Adaptation and Piggybacking. Our system provides their generalized applicability by modelling context recognition applications using a state machine abstraction. Consequently the resulting applications are structured as combinations of states and transitions. To aid rapid prototyping our system is equipped with an off-line development tool which allows the creation and (code) generation of state machines using a graphical editor. We evaluate our system by creating a test application. Using a precise hardware set up, we perform energy measurements to demonstrate the energy savings of these different techniques.**

***Keywords*-Context; System; Energy; Smart Phones**

I. Introduction

Intelligent environments provide useful services to its users by identifying the tasks which users are currently involved in. Examples could be an environment which automatically adjusts the intensity of lights in the offices when a user is working or an environment which guides users to park their cars in the parking places. For either of these or many scenarios, intelligent environments rely on the context information of the user i.e what the user is currently doing and/or where the user is at the moment. Among various platforms for determining user′s context, smart phones are one of the most widely used platforms.

There exist many context recognition systems and applications such as [1], [2], [3] for smart phones. These and similar systems provide recognition of their desired context features with suitable accuracy. However, despite the fact that modern day smart phones have become more resourceful in terms of computation power they still suffer from the lack of breakthrough advancements in the domain of battery capacity. As context recognition applications require energy consuming operations, their use can have significant effect on the energy usage of phones. This, in turn, can lead to less time for other basic operations such as talk time, for instance.

To overcome this limitation there exist systems which provide energy efficient solutions for context recognition using smart phones. However, most existing approaches are usually specific to particular context types, e.g. applications targeting localization solutions [4] focus on achieving localization in energy efficient manner, applications targeting activity recognition [5] specialize on how to recognize certain activities in an energy efficient manner, etc. Consequently, there exist a number of energy efficiency techniques which are used for specific contexts but despite of having the potential, are not utilized for other context domains.

To bridge this gap, we present an energy efficient context recognition system for smart phones which provides generalized applicability of four energy efficiency techniques for location sensing described in [4]. These four techniques include Suppression, Substitution, Piggybacking and Adaptation. With our system these four techniques can be applied to any type of context. Our system achieves this generalized applicability by modelling applications using a state machine abstraction. Consequently, applications are structured as combination of states and transitions. A state in our system refers to some context characteristic that an application computes at different stages of its execution. In order to compute the context characteristics our system relies on an existing component system for context recognition [6]. A transition in our system refers to conditional context changes between the states. The transitions are realized as rules (i.e. conditional expressions) and the system uses a rule engine to automatically evaluate the transitions that initiate state changes.

Our system is equipped with a visual graphical editor implemented as Eclipse [7] plug-in. The editor allows developers to create state machines required for their applications. It can also generate the source code of the designed state machines which can then be directly used in an application, thereby relieving developers from coding development effort.

We evaluate our system experimentally with a test application. Using a precise hardware set up similar to the one shown in [8], we perform energy measurements on the application and demonstrate the potential energy savings of the different energy efficiency techniques.

The remainder of this paper is structured as follows. Next we introduce the four energy efficiency techniques to make the paper self-contained. In Section III, we describe the design rationale behind our system. In Section IV, we describe our system in detail. In Section V, we discuss

some implementation details and in Section VI, we present an evaluation of the system. In Section VII, we discuss the related work and conclude the paper in Section VIII.

II. Energy Efficiency Techniques

For completeness, we briefly outline the four energy efficiency techniques introduced in [4]. These techniques have been implemented for Android phones in the context of location sensing scenarios. The evaluation detailed in [4] shows considerable energy savings and therefore provides compelling grounds for having a system which can use these techniques in a generic manner - that is beyond the scope of location sensing.

- *Suppression:* Suppression refers to the use of low energy sensors to first determine the user movement and then use high energy sensors to perform localization.
- *Substitution:* Substitution refers to the use of low energy consuming location sensing sources instead of high energy consuming source, provided the accuracy of low energy consuming source is equal or better than the accuracy of high energy consuming source.
- *Adaptation:* Adaptation refers to the adjustment of location sampling parameters based on the current battery levels or other user settings. If the battery level is below a certain threshold then the location sensing time interval or location sensing distance threshold is increased.
- *Piggybacking:* Piggybacking refers to combining multiple location sensing request into a single request. If there are more than one location based applications asking for the current location, piggybacking uses the sensing parameters of one application which are finer than the sensing parameters of all other applications. It then reports the location information to all the applications based on the chosen parameters.

III. Design Rationale

In order to provide generic applicability of the previously mentioned energy efficiency techniques beyond location sensing, a generic system for energy efficient context recognition must provide support for the following three functions:

- *Conditional execution:* In order to enable *Suppression*, a system must be able to recognize context in multiple steps (possibly structured in a hierarchical fashion) such that low energy consuming operations are performed first to decide whether high energy consuming operations are required. This can be done by enabling the conditional execution of different steps based on the context that has been recognized so far.
- *Adaptation support:* To enable *Substitution* and *Adaptation*, a system must provide adaptation support so that an application can modify the behavior of its context recognition logic by adjusting settings. This can be triggered either based on the cost for determining context (Substituion) or on the state of the device, e.g. its remaining battery power (Adaptation).
- *Request multiplexing:* To enable *Piggybacking*, a system must be able to capture the requests from multiple applications and handle them jointly in order to avoid duplicate sampling and processing.

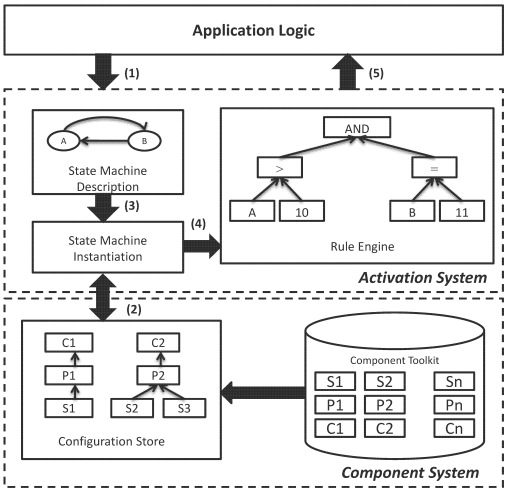

Figure 1. Architectural Building Blocks

In order to support these functions, we use state machine abstraction as basis for our context recognition system. Since a state machine introduces states and (conditional) transitions, they are a simple and flexible way to model the conditional execution. For example, developers can structure applications such that states represent low and high energy consuming operations and transitions represent the rules for switching between them. Similarly, to achieve adaptation support, developers can use states to represent alternate context recognition methods with different energy and accuracy cost to recognize same context or use them to represent different application settings relevant to particular battery levels. The transitions then provide switching between different recognition methods or different application settings. In addition, adaptation support can also be achieved by structuring applications as combination of multiple state machines instead of just one state machine. In this case, the transitions provide switching between different state machines. Finally, to support request multiplexing, we rely on an existing component system described in [6] as execution platform for the state machines. This system models individual context recognition actions as parameterizable component configurations that can be executed by a common runtime system. To enable piggybacking in a generic fashion, the system applies configuration folding which dynamically analyzes a set of simultaneously executed configurations in order to eliminate redundant sampling and computations.

IV. System Architecture

The building blocks of the resulting system are depicted in Figure 1. The system consists of two subsystems namely the activation system and the component system. The activation system is responsible for managing the state

machines that define the context recognition steps of different applications. Underneath the activation system, the component system provides generic context recognition support realized by configurations of components as described in [6].

The main architectural building blocks of activation system are the state machine description and the associated runtime system which executes them. The runtime system consists of the state machine instantiation logic and the rule engine. In the following, we first describe the basics of the state machine model before we discuss the interaction of the different building blocks at runtime.

A. State Machines

The state machine model introduced by the activation system consists of two elements, namely states and transitions. Conceptually, a state represents a step in a context recognition application at an arbitrary level of granularity. During this step, the application continuously recognizes certain context characteristics using a particular sensing and processing logic until a certain context constellation is detected. The constellations that are relevant for the application are represented as transitions consisting of a source state, a target state and a rule, i.e. a set of conditions that describe the applicable constellations. When a constellation associated with a transition (whose source state is the current state) is detected, the state machine moves to the target state, thereby, effectively moving on to the next step in the application which may then modify or replace the sensing and processing logic. In the following, we discuss both states and transitions in more detail.

1) States: As indicated previously, a state represents a particular step in an application that is associated with a particular sensing and processing logic. To model this logic, states are associated with a collection of component configurations that are used to determine the relevant context characteristics. Depending upon the granularity of the step, the attached configurations could be detecting range of context characteristics with different levels of details. As an example, Figure 2 depicts a state machine with two states. These states capture user's mode of transportation. If the user is travelling in train, the state machine remains in the Travelling in Train state otherwise it switches to the Waiting for Train state.

2) Transitions: A transition represents a conditional change between two states. In our system we realize the conditions using rules. Every transition is associated with one or more configurations attached to its source state. Each transition is associated with one rule, however, a rule may consist of one or more than one conditions. Each condition is attached to only one configuration and it is modelled as an abstract syntax tree to enable the formation of composed conditions. Each condition consist of three nodes namely the operand, the operator and the value. If there are more than one conditions in a rule they can be joined together with AND or OR clause. For a transition between states to take place it is necessary that the associated rule is evaluated to be true.

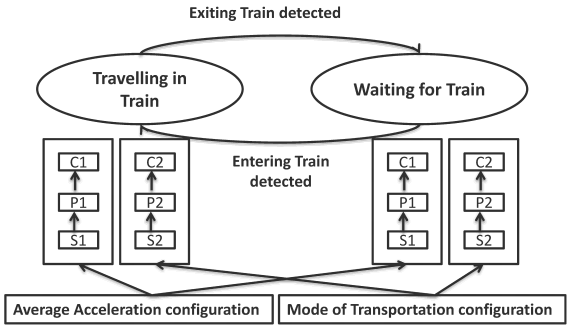

Figure 2. Example of state machine with two states

- Operand: An operand receives the outcome of the configuration associated with the condition. A rule is evaluated every time the operand receives any update.
- Operator: An operator represents a mathematical operator for evaluating the operand. The operators supported by our system include GREATER_ THAN (if operand is greater than the value), LESS_ THAN (if operand is less than the value) and EQUALS_ TO (if operand is equals to the value).
- Value: A value is used to compare the output of the operand using the operator. A value can be either of NUMERIC data type or character based data type such as STRING.

Examples of rules for the state machine shown in Figure 2 are shown in Figure 3. The rules for Figure 2 assumes that there are two configurations "Average Acceleration" (which uses accelerometer data to determine acceleration and body posture of the user) and "Mode of Transportation" (which uses WiFi and sound information to determine the type of vehicle) consisting of different components and attached to both states "Travelling in Train" and "Waiting for Train". The condition tree for the rules attached to the transitions of both states has seven nodes, three nodes for representing two conditions and one node for the aggregation of the two conditions.

B. Interaction

To demonstrate the interaction between the architectural building blocks, we briefly outline the general process depicted in Figure 1: (1) As the first step the application passes a state machine description to the activation system. According to the model described previously, a state machine description consists of configurations attached to different states and rules associated with transitions between different states. (2) Once the state machine description has been passed, the activation system identifies the starting state in the state machine and instantiates that state by sending the request to the underlying component system. The component system in turn instantiates the components and links described in the configurations attached to the state. Once the configurations are instantiated they start determining the context characteristics. Upon completion of each recognition cycle i.e from sampling of sensor data to processing of data and from processing of data

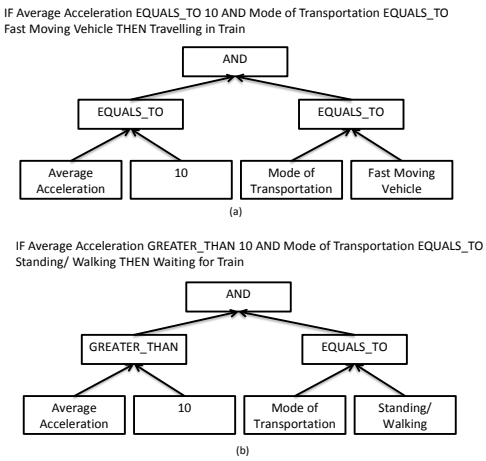

Figure 3. Examples of condition trees for two rules

to classification of processed data, the configuration sends the output to the activation system. (3) Next, the activation system instantiates the rules associated with the current state and then (4) activates the rule engine by passing rules to it. The rule engine evaluates the rules based on the output send by the configurations and when a rule is evaluated to be true a change of state takes place. (5) When the need for a state change is detected, the activation system signals the change of state to the application. At the same time the activation system stops the configurations attached to the current state, instantiates the configurations associated with the new state and instantiates the associated rules. The rule engine then starts evaluating the new rules using the new configurations and the process continues.

C. Discussion

As depicted in Figure 4, in this section we discuss how the proposed system architecture enables the four techniques to be used not just for location-based application but for any type of context recognition application.

- *Suppression:* As any type of configuration (for determination of movement, for determination of sound etc.) can be attached to a state, it is completely flexible to use Suppression irrespective of type of target context. An example of Suppression could be to use audio sensor at low sampling rate to determine presence of some particular sound e.g. sound of opening of door first and then turn on GPS for outdoor location sensing. With our system this can be realized by having two states, one with configuration for determining the sound and other for determining the location. This shows that with state machine abstraction employed in our activation system, we can enable Suppression for different contexts.
- *Substitution:* Like Suppression the Substitution can also be applied in a generic way by the activation system. As Substitution means replacement of one sensing method with another, we can have different configurations for different sensing methods attached to different states. We can also have different state machines to achieve the same effect. With the evaluation of rules attached to transitions, the activation system can substitute one sensing method with other. e.g we can use one configuration for GSM localization attached to one state and second configuration for GPS localization attached to other state.
- *Adaptation:* Just like Suppression and Substitution, adaptation can conceptually be realized by introducing separate states for the different adaptation levels (e.g. one state for detecting movement if the phone's battery level is low and another state for detecting movement if the phone's battery level is high) and associating different configurations (e.g. one with a low sampling rate and one with a high sampling rate) with the states. However, since this will generally lead to complex state machines with a large number of transitions to represent the adaptation, it is typically more convenient to use separate machines to represent the different levels and to introduce an additional "meta" state machine to detect the appropriate adaptation level. This meta state machine is then responsible for activating the appropriate variant of the actual state machine that performs the recognition. In order to thoroughly support this, our implementation of the system described in the next section enables developers to directly associate state machines with states which technically enable a hierarchical composition. Yet, it is noteworthy to point out that this is primarily a tool for simplifying the development as it is possible to generate a single machine to handle adaptation.
- *Piggybacking:* As mentioned in Section III, the underlying component system that we use to run the sampling and processing logic for the activation system already provides configuration folding as a generic solution to achieve Piggybacking. Thus, if a state of a state machine exhibits multiple configurations or if multiple state machines are executed simultaneously, the configuration folding algorithm of the component system will ensure the energy efficient execution of the complete set of configurations that is running at the same time. More details on this including an evaluation of the possible savings can be found in [6].

Figure 4 illustrates the generic applicability of these techniques which, for the sake of clarity, uses a "meta" state machine to control the execution of two other state machines. Both the states in the meta state machine have a variant of configuration A attached to it. Depending upon the transition, either state machine 1 or state machine 2 is active at any time. This switching of state machines demonstrates application of Adaptation. In state machine 1 and state machine 2, there are three states and depending upon the definition of the states, the switching between states can either represent Suppression or Substitution. Lastly, State 1 and State 2 in the two state machines have two configurations attached to them. Thus, configuration folding is applied to these configurations in the two states

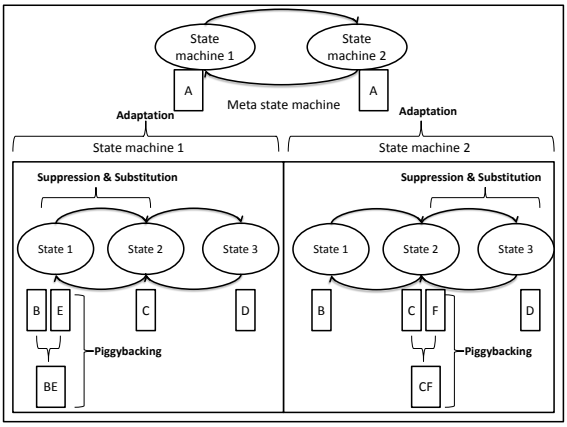

Figure 4. Generic applicability of four energy efficiency techniques

to produce a single configuration which demonstrates the applicability of Piggybacking.

V. Implementation

To evaluate the system, we have implemented the activation system and integrated it with our existing component system. In addition, to simplify the application development using the proposed abstraction, we have created a visual development tool that is capable of hiding the low level details by means of code generation. In the following, we briefly describe the implementation of both.

A. Activation System

The activation system has been implemented for Android. The system is implemented as an Android service and applications can start and stop state machines by sending their description as an Andorid Parcelable to the activation system service. The activation system service identifies the default state and if the Parcelable also contains a meta state machine, it first identifies the default state machine and then the default state. Once the state is identified the service starts another service that runs the component system and passes it the configurations attached to the state. The component system service performs configuration folding and instantiates the configurations. The outputs of running configurations are signaled via Android broadcasts.

Once the configurations are started, the activation system service starts the rule engine service. The rule engine service registers the broadcast receivers for the currently executed configurations. When the broadcasts from the configurations are received, the rule engine evaluates the rules for all transitions of the current state. If a state change occurs, the activation system service modifies the set of running configurations by starting and stopping them according to the state machine description and updates the rule engine and the registration of broadcast receivers accordingly. Finally, the change in states is sent to the applications via broadcasts so that it can perform any desired task.

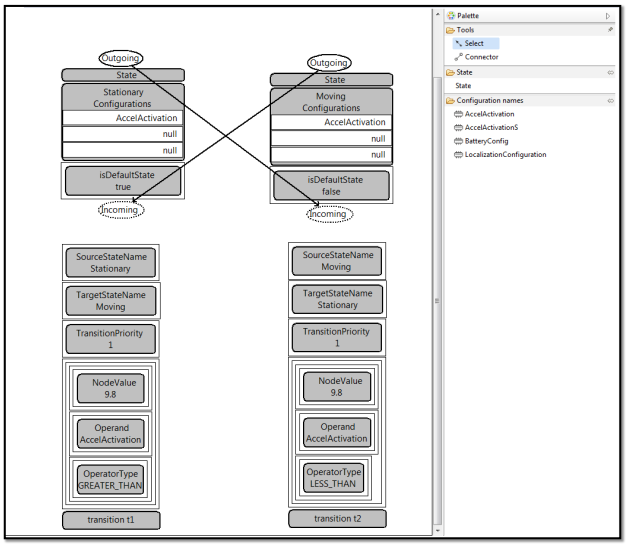

Figure 5. Screenshot of the Visual Editor

B. Visual Editor

Depending on the size of the state machine, the number of lines of code required to define a state machine description can be comparatively large. For example, a simple state machine consisting of two states with one configuration attached to each state and one outgoing transition per state already requires more than 100 lines of code. Although, this code is conceptually trivial, due to the sheer size and the fact that the definition requires the repetitive and consistent use of different model elements such as states, transitions, nodes, operands, operators, values, etc. its manual creation is a tedious process that is prone to errors.

To mitigate this, we have created an Eclipse-based graphical editor plug-in for the visual definition of state machines. The editor allows developers to create and modify state machine by simply dragging, dropping and connecting states. The editor supports the attachment of configurations to the states and also allows the definition of rules for the transitions. Furthermore, the editor provides a state machine validation mechanism with which it can identify missing or inconsistent details in the state machine description. Lastly, the editor also supports code generation for the designed state machine. Figure 5 and Figure 6 show a screenshot of the editor with a two state machines and a fragment of the resulting generated code, respectively.

VI. Evaluation

In order to demonstrate the effectiveness of our system we created a test application for performing indoor localization by using our system. The application was realized using different configurations which are detailed in the next section. The developed application demonstrates the use of different targeted energy efficiency techniques as has been depicted in Figure 4. It is worth mentioning that we have chosen an indoor localization application primarily for illustrative purposes, since the various context

```
StateMachineConfiguration configuration = new
                    StateMachineConfiguration(configurationName);
// create states configurations                                   Defining State
State state0 = new State();
state0.setName("Stationary");
state0.setDefaultState(true);
AccelActivation AccelActivation0 = new AccelActivation();
state0.setConfigurations(AccelActivation0.createConfiguration("AccelActivation"));
configuration.setState(state0);
// create transition configurations                               Defining Transition
Transition transition0 = new Transition();
transition0.setName("transition t1");
transition0.setPriority(1);
transition0.setSourceState("Stationary");
transition0.setTargetState("Moving");
Rule rule0 = new Rule();                                          Defining Rule
rule0.setName("Rule");
ConditionTree conditionTree0 = new ConditionTree();
//setting up value
Value value0 = new Value();
value0.setValue("9.8");
//setting up operand
transition0.addConfigurationsAttacdhedToTransition("AccelActivation");
Operand operand0 = new Operand(Platform.ANDROID,"AccelActivation");
//setting up operator
Node node_operand0 = new Node(NodeType.OPERAND,null,null,null,operand0,null,null);
Node node_value0 = new Node(NodeType.VALUE,null,null,null,null,value0,null);
Node node_operator0= new
Node(NodeType.OPERATOR,node_operand0,node_value0,OperatorType.GREATER_THAN,null,null,null);
//setting up nodes in the condition tree
conditionTree0.setCondArrayList(node_operand0);
conditionTree0.setCondArrayList(node_operator0);
conditionTree0.setCondArrayList(node_value0);
```

Figure 6. Generated code snippet of a state machine

recognition approaches are easy to explain and understand. The basic methodology and principles for integrating the four energy efficiency techniques in the form of certain state machine configurations, however, can be applied to any kind of application which supports the recognition of context via alternative approaches.

A. Application

The indoor localization application was built in two stages. In the first stage, we created the component configurations required for performing the localization. In the second stage we created the state machines to arrange those configurations such that the Suppression, Substitution and Adaptation are used. As [6] already provides a detailed evaluation on Piggybacking, we skip its evaluation using the test application. The configurations were created using the existing graphical editor and the component toolkit provided by the component system [6]. In total we created four configurations which include a configuration for measuring the battery status of an Android device (BatteryConfiguration) as basis for adaptation, a configuration to determine user movement using the device's accelerometer (AccelConfiguration), a configuration to determine user movement using the audio sensor (SoundConfiguration) and a configuration for performing the actual localization (LocalizationConfiguration). These configurations were then used to create different state machines using the graphical editor and its code generation utility for state machines presented in this paper.

In order to demonstrate Adaptation we created two state machines to perform the localization (using different strategies) and a meta state machine to control the adaptation process. Thus, the meta state machine is used to switch between two state machines each representing different settings for the localization. To do this, the meta state machine used BatteryConfiguration in two states to start/stop one of the state machine. The rule used by the meta state machine was to use state machine 1 when the phone's battery is more than 50% and use state machine 2 when the phone's battery is less than 50%.

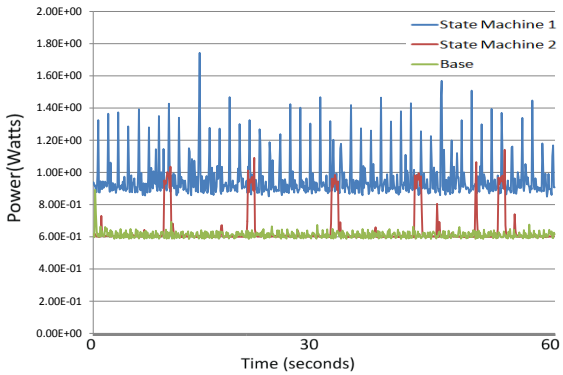

Figure 7. Power consumption when Adaptation is used. State machine 1 perform continuous WiFi scans and State machine 2 performs WiFi scans every 10 seconds.

In one of the two state machines, the LocalizationConfiguration was parametrized to perform continuous WiFi scans whereas for the other state machine the LocalizationConfiguration was parametrized to perform WiFi scans every 10 seconds. The time interval of 10 seconds is chosen to highlight the energy savings compared with continuous location recognition. To demonstrate Suppression, we added a second state to the state machines that used the AccelConfiguration in order to suppress the use of the LocalizationConfiguration in cases where the device is not moved (i.e. accelerated). To demonstrate Substitution, we copied the state machine and replaced the AccelConfiguration to determine the user movement with the SoundConfiguration. Though, accelerometer data is typically used for the determination of movement in scenarios where user is moving between places a distinct change of ambient noise can be an alternate source of information.

B. Experiments

For our experiments with the application, we used a Samsung Galaxy Nexus phone running on Android 4.2 as our target platform. To determine the energy usage, we used a precise energy measurement hardware set-up similar to one described in [8] for measuring the power consumption. We first computed the base power drain with device's screen brightness set to minimum (0.609 watts). Thereafter, we performed additional measurements showcasing the three techniques. For Adaptation, the power consumption of the first state machine in which the LocalizationConfiguration was parametrized to perform continuous WiFi scans was computed to be 0.96 Watts where as for the state machine in which the LocalizationConfiguration was parametrized to perform WiFi scans every 10 seconds was computed to be 0.634 Watts. In both cases the state machines were executed for 60 seconds. We subtracted base power consumption to obtain actual power consumptions. For first state machine the actual power consumption was calculated to be 0.96-0.609 =

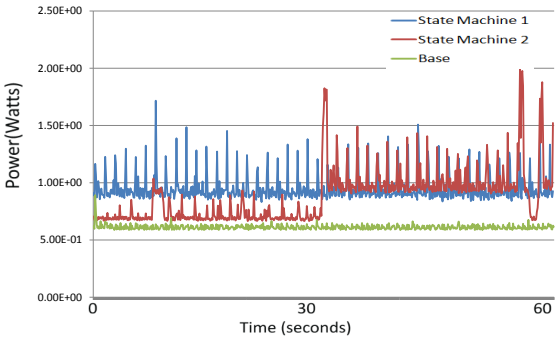

Figure 8. Power consumption when Suppression is used. State machine 1 is attached to AccelConfiguration and State machine 2 is attached to AccelActivation and LocalizationConfiguration both.

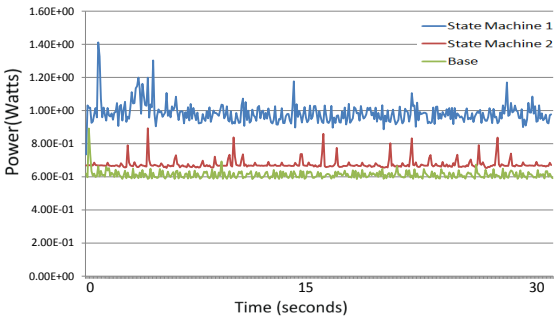

Figure 9. Power consumption when Substitution is used. State machine 1 is attached to SoundConfiguration and State machine 2 is attached to AccelConfiguration

0.351 Watts where as for the second state machine it was 0.634-0.609 = 0.0252 Watts. The net power saving for executing the localization with different setting were computed to be(1-0.0252/0.351)*100 = 92%. The power consumption graphs for Adaptation are shown in Figure 7.

For Suppression, we assumed that user does not move for half of the execution time of the application i.e. for 30 seconds out of 60 seconds (similar to experiment settings in [4]), the user remains static. As a result the state machine used for Suppression remains in state attached with AccelActivation Configuration for 30 seconds and switches to other state attached with AccelActivation and LocalizationConfiguration both for the remaining 30 seconds. The average power consumption for this state machine was computed to be 0.842 Watts. After subtracting the base power the actual power consumption becomes 0.842-0.609 = 0.233 Watts. In order to measure the power savings for Suppression, we computed power consumption of state machine continuously executing LocalizationConfiguration. Its power consumption was computed to be 0.946-0.609 = 0.337 Watts. Hence, the net power savings for Suppression were computed to be (1-0.233/0.337)*100 = 30.8%. The power consumption graph for Suppression is shown in Figure8. We can see that the power consumption of state machine 1 is almost same through out the time since it is continuously performing localization. The power consumption of state machine 2 remains low for about period of 30 seconds. This is due to the fact that user is not moving. Afterwards, as user starts to move, the localization is performed and hence the power consumption is increased. It can be seen in the figure that during this time period the power consumption of state machine 2 is higher than state machine 1. This is due to the fact that during this period state 2 of state machine 2 is executing LcalizationConfiguration and also AccelConfiguration.

For Substitution, the actual power consumption for state machine with SoundConfiguration was computed to be 0.982-0.609 = 0.373 Watts and for state machine with AccelConfiguration was computed to be 0.675-0.609 = 0.0662 Watts. The net power savings if AccelConfiguration is used instead of SoundConfiguration were (1-0.0662/0.373)*100 = 82.2%. The power consumption graph for Substitution is shown in Figure 9.

As mentioned earlier and also demonstrated by the test application, it can be seen that our system can be used to develop applications targeting any context type such that they can benefit from the supported energy efficiency techniques irrespective of their target context.

VII. Related Work

Existing approaches for achieving energy efficient context recognition for smart phones provide solutions either at the application level or at the system level. At application level, the provided solutions are specific to the context type which the application targets and thus are useful to the application only. At the system level, the provided solutions can be utilized by different applications targeting different contexts. Some of application level solutions include [1],[9]. [1] provides a microphone based sound classification approach for Apple iPhone. The approach suggests to process only those sound samples which have certain energy level to avoid processing of non important ones. [9] uses a hierarchical recognition scheme with variable step size and adjustment of sliding window size for accelerometer data for recognizing human activities such as Standing, Sitting, Walking etc.

Examples at the system level include [10],[11],[12]. [10] uses sensing pipelines for accelerometer, microphone and GPS sensors. These pipelines provide adaptive processing of microphone and accelerometer data when data quality is low. Depending upon the mobility and behaviour pattern of user, it also provides intelligible triggering of power hungry GPS sensor. [11] provides energy savings of several applications by reducing the amount of sampling and processing. The system allow developers to specify context recognition requirements for their applications. Based on the requirements the system uses minimal sensors and computations to determine the desired context. [12] uses inferring mechanisms instead of continuous sensing to determine current context. It provides inference by learning relationships between different context attributes.

The system presented in this paper provides generic applicability of Substitution, Suppression, Adaptation and Piggybacking techniques described for localization in [4]. Comparing the activation system with above mentioned application level and system level solution indicates that

activation system is complementary to these solutions. To mention, these four techniques have also been discussed in systems targeting contexts other than location. Examples include [13],[14],[15],[16]. However, unlike our system which provides generic applicability of all four techniques, these systems uses only a subset of them.

The presented activation system uses a state machine abstraction. The state machine abstractions have been used in existing context and activity recognition systems. Examples include [13],[17],[18],[19]. [17] uses state machine abstraction for the determination of suspicious human activities based on video surveillance. [18] uses a multi-layered state machine for human activity recognition and object identification. [19] uses finite state machines for detection of common household activities such as cooking, eating, brushing teeth etc. The system observes location of users by video cameras fitted on the ceilings and their actions with the help of a wearable sensors placed on users arms. These example systems show that the applicability of state machines in context and activity recognition is well established in the existing work.

VIII. Conclusion

Existing solutions for energy efficient context recognition for smart phones are usually fine tuned for specific context types. As a result their applicability to other contexts is limited. To bridge this gap, in this paper we have presented a generic energy efficient context recognition system for smart phones. The system provides generalized applicability of four energy efficiency techniques namely Suppression, Substitution, Adaptation and Piggybacking for any context type. Our system achieves this by using state machine abstraction for the modelling and execution of context recognition applications. We evaluated our system by creating a test application and measurements showed energy savings when the test application uses the aforementioned techniques supported by our system.

IX. Acknowledgements

This work is supported by UBICITEC e.V. (European Center for Ubiquitous Technologies and Smart Cities) and GAMBAS (Generic Adaptive Middleware for Behavior-driven Autonomous Services) funded by the European Commission under FP7 with contract FP7-2011-7-287661.

References

[1] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "Soundsense: Scalable sound sensing for people-centric applications on mobile phones," in *MobiSys 2009*.

[2] M. U. Iqbal, N. Fet, S. Wagner, M. Handte, and P. J. Marrón, "Living++: A platform for assisted living applications," ser. UbiComp '13 Adjunct, 2013.

[3] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *SenSys 2008*.

[4] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10, 2010.

[5] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer, "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach," in *Wearable Computers (ISWC)*, 2012.

[6] M. Iqbal, M. Handte, S. Wagner, W. Apolinarski, and P. Marron, "Enabling energy-efficient context recognition with configuration folding," in *PerCom 2012*.

[7] "Eclipse," http://www.eclipse.org/.

[8] M. Iqbal, M. Handte, S. Wagner, W. Apolinarski, and P. Marron, "Configuration folding: An energy efficient technique for context recognition," in *PerCom 2012 (Workshops)*.

[9] Y. Liang, X. Zhou, Z. Yu, B. Guo, and Y. Yang, "Energy efficient activity recognition based on low resolution accelerometer in smart phones," in *Proceedings of the 7th International Conference on Advances in Grid and Pervasive Computing*, 2012.

[10] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The jigsaw continuous sensing engine for mobile phone applications," in *SenSys 2010*.

[11] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song, "Seemon: Scalable and energy-efficient context monitoring framework for sensor-rich mobile environments," in *MobiSys 2008*.

[12] S. Nath, "Ace: Exploiting correlation for energy-efficient and continuous context sensing," *IEEE Transactions on Mobile Computing*, 2013.

[13] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," in *MobiSys 2009*.

[14] A. Kansal, S. Saponas, A. B. Brush, K. S. McKinley, T. Mytkowicz, and R. Ziola, "The latency, accuracy, and battery (lab) abstraction: Programmer productivity and energy efficiency for continuous mobile context sensing," in *Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages ; Applications*, 2013.

[15] T. Park, J. Lee, I. Hwang, C. Yoo, L. Nachman, and J. Song, "E-gesture: A collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices," in *SenSys 2011*.

[16] S. Kang, Y. Lee, C. Min, Y. Ju, T. Park, J. Lee, Y. Rhee, and J. Song, "Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments," in *PerCom 2010*.

[17] A. Fernandez-Caballero, J. C. Castillo, and J. M. Rodrguez-Snchez, "Human activity monitoring by local and global finite state machines," *Expert Systems with Applications*, 2012.

[18] D. Mahajan, N. Kwatra, S. Jain, P. Kalra, and S. Banerjee, "A framework for activity recognition and detection of unusual activities." ICVGIP, 2004.

[19] T. Teixeira, D. Jung, G. Dublon, and A. Savvides, "Recognizing activities from context and arm pose using finite state machines," in *Distributed Smart Cameras, ICDSC 2009.*