International Journal of Computer Science Engineering and Information Technology Research
Vol.1, Issue.1 (2011) 40-71

INTELLIGENT WIDGET RECONFIGURATION FOR MOBILE PHONES

Peter K.K. Loh, C.T. Lau and B.H. Chan
Nanyang Technological University, Singapore

ABSTRACT

A significant amount of research work in user interface design exists with a proportion of this extendable to mobile phone platforms. Some investigate the effect of user ability on interface generation for mobile applications. Other works analyzed how different contexts and mobile platforms affect the generation of these interfaces. However, most of these existing works require a significant degree of context requirements modeling before interface reconfiguration takes place. Few on-the-fly reconfiguration approaches exist that learn from user interactions as well as contextual information received by a mobile phone. With the explosive growth of new applications for the mobile phone, its user interface is quickly becoming flooded with application widgets. This work investigates some on-the-fly approaches that learn and formulate rules from user interactions and contextual information received by the mobile phone. Performance evaluations demonstrate how a simple neural network-based engine is able to improve the prediction accuracy of the interface reconfiguration in a mobile phone.

Keywords: mobile phone, context-aware, intelligent interface, widget reconfiguration, neural network, rules

INTRODUCTION

People are becoming increasingly dependent on mobile information devices. With increased pervasiveness of wireless connectivity and technology

advancements, the smart mobile phone is progressively taking on more important roles to process, collate and delegate information. A contemporary mobile handset typically comes with many integrated features, which previously were available only on desktop PCs or laptops such as internet access, email, word processing, and video viewing. An increasing number of mobile phones are also equipped with additional hardware like sensors to extend its capabilities e.g. the accelerometer in the Samsung Blade S5600v [31], the motion recognition sensor in the Samsung S310 [29], the proximity sensor in the HTC Touch Pro 2 [25], and the Nokia 5500's tilt sensor that supports novel game genres [28]. Together with better processing power, these mobile phones have become mini multimedia computers proffering support for an increasing spectrum of new applications and features.

However, these technological enhancements to a mobile phone also herald a new set of user problems. As the number of supported widgets increases, widget management becomes increasingly complex. Locating relevant or interesting widgets becomes a chore as the user interface gets cluttered with irrelevant widgets [17]. In a recent study [22], most new mobile phone owners indicated that they were adverse to using new services that were confusing or difficult to access. Contemporary mobile phones address these problems partially via screen organization tools like window managers, widget toolbars [31] or multiple home screen pages or *scenes* [23]. Usage of these tools requires proficiency with the mobile phone's key controls to be able to correctly and efficiently re-organize the screen's application widgets. For example, in the HTC Hero [24], each scene must be pre-specified by the user with appropriate widgets for different contexts e.g. work, travel, social etc. and are non-adaptive.

When changes occur over time in a mobile phone user's lifestyle and/or roles, new applications may need to be downloaded, existing applications may be upgraded or older applications rendered obsolete. In such cases, manual widget re-organization would have to be performed repeatedly with the above tools. This can be both tedious and time consuming.

From a mobile phone user's perspective, some degree of intelligent widget control should be provided as one goes about his/her daily activities. For instance, a user driving to work will more likely use the phone in "hands-free" mode for voice communications instead of SMS or email. A student who normally uses the phone for entertainment and social interaction will likely not use it during curriculum time. It is also less likely that an employee would want to invoke a game application or view a movie at the work place than when travelling via public transport or resting at home in the evening. Similarly, a mobile business executive or tourist may place more emphasis on GPS and location-based applications and functionality than an office-bound employee, albeit using different categories of services. We ask the following questions:

- Does a user need all or even most of a mobile phone's capability?
- Does a user need the same subset of applications and functionality all the time?
- Can a mobile phone be made to learn and recognize a user's context?

It is quite apparent that different subsets of a mobile phone's capabilities appeal to different users depending on roles/interests. On a regular basis, many users also make use of specific applications and functionality in a fairly deterministic pattern depending on context. A typical mobile phone user's context may be defined in terms of usage pattern, date, time of day, and location as a basis. With the aid of suitable sensor inputs, additional contextual information may be gleaned e.g. how far has the user moved from the previous location, how fast is the user moving now, heart rate (stress level) etc. However, at the time of this paper, there is no reported work that offers a learning engine with dynamic context-aware reconfiguration of a mobile phone interface. As a consequence, mobile phone users would have to constantly navigate through and manually reconfigure a complex and confusing set of excess widgets that they either do not need or no longer use.

MOTIVATION AND CHALLENGES

As mobile phones become a necessity to the daily lives of most people, there is a need to find an effective solution to the issues highlighted previously. As the

population is also graying in several countries, there may be more users who will encounter difficulties with the interfaces of contemporary mobile phones [15].

Currently, there is no reported mobile phone engine that is able to learn and automatically reconfigure the interface widgets based on a user's context. As an example, a user's context could be defined by time, location, traffic condition, phone usage pattern and even personal biometrics like heart/perspiration rate etc. Currently, widgets will reside on the mobile phone interface even if the user has not used the applications for an extended period of time. These widgets will remain until the user specifically removes the application. Similarly, when new applications are installed, the user can either manually organize the corresponding widgets or just allow the widget "clutter" to get worse. Reasons for the lack of such capability could include anticipated high processing overheads, development constraints and potential increased access complexity introduced by such engines [18].

With an efficient context-aware reconfiguration engine, however, users need not expend considerable effort to navigate and locate a particular application as the system will automatically select, retrieve and display the icons most likely to be used for that user. An intelligent reconfiguration engine will also learn from the perceived context and suitably adapt the interface for the user [18] doing away with frequent manual reconfiguration. Challenges facing the development of such an engine would then include:

- Need for a simple and efficient design that would not introduce unnecessary overheads or more complexity to the user interface.
- The engine must be transparent to the user while working in tandem with the operating system of the mobile phone
- The engine must learn and automatically adapt the interface according to the specified context

This paper proposes an intelligent, context-aware interface reconfiguration engine prototype for the mobile phone called *SmartIFace* and is organized as follows. Section 1 provides the background of the existing situation

and associated problems while Section 2 presents the motivation and challenges behind the research. Section 3 presents a review of related research. Design details of the system architecture and learning engine are in Sections 4 and 5. Section 6 presents the system test results as well as a performance analysis and evaluation. The paper concludes with Section 7 followed by the references.

REVIEW OF EXISTING RESEARCH

The following systems, Supple, Gaze-X, Dynamo-AID, Situations and MIMIC, make use of contextual information that is typically user-specified, to generate user interfaces. Some of these interfaces can be exported to mobile phone platforms via additional mechanisms.

SUPPLE

Supple [11] automatically generates user interfaces according to the user's physical ability, device and usage. Supple is able to create personalized interfaces better suited to the contexts of individual users. Users with physical limitations and impairments form a particular group targeted by Supple. A subsystem called Ability Modeler performs a one-time assessment of a person's motor ability and builds a model of those abilities. The result is used during interface generation. These automatically generated interfaces greatly improve the speed, accuracy and satisfaction of users with motor impairments.

A subsystem called Arnauld is also used to obtain user responses to generate a cost function that closely approximates the desired behavior. An optimization algorithm determines the user interface that satisfies the platform device's constraints while minimizing the cost function. By supplying different device constraints and cost functions, different styles of user interfaces may be produced. However, Supple's functionality is currently restricted to window-based interfaces found on desktop platforms. Although Supple is written in Java, it is currently unable to run on a mobile phone as it uses libraries from Java SE.

GAZE-X

Gaze-X [16] is an agent-based intelligent system that supports multimodal human-computer interaction. The system comprises 2 main parts – context modeling and context sensing. It models user's actions and emotions and then adapts the interface to support the user in his activities. The context information, known as W5+ (who, where, what, when, why, how), is obtained through a number of human communication modalities, such as speech, eye gaze direction, face and facial expression, and a number of standard interface modalities like mouse moves, keystrokes and active software identification. Various commercially available solutions such as voice recognition software, image processing software are required for the multimodal inputs.

The inference engine used is based on case-base reasoning, a type of lazy learning method [3]. Lazy learning methods store the current input data and postpone the generalization of data until an explicit request is made. The case-base used is a dynamic, incrementally self-organizing, event-content-addressable memory that allows facts retrieval and events evaluation based on user preferences and generalizations formed from prior inputs. Based on the evaluation, Gaze-X will execute the most appropriate user-supportive action. The case-based reasoning can also unlearn actions according to user instructions and thereby increasing its expertise in user-profiled, user-supportive, intelligent interaction.

Gaze-X runs in either unsupervised or supervised modes. In the unsupervised mode, the user's affective state is used to decide on his satisfaction with the executed action and adaptive, user-supportive actions are executed one at a time. In the supervised mode, the user explicitly confirms that a preferred action has been executed and may provide feedback to the system. Gaze-X has to be setup initially in the supervised mode to build up the profile of the user using the system. Once the system has captured enough cases of the user, the system would then be able to operate correctly in the unsupervised mode. Gaze

X currently runs only on desk-top platforms based on Linux, Windows, or Mac Os X.

DYNAMO-AID

Dynamo-AID (Dynamic Model-Based User Interface Development) [4-5] is a design process that includes a proposed runtime architecture to develop user interfaces for context-aware systems. However, the designer must first construct the declarative models which specify the interactions. Next, the models are serialized to an XML-based high-level user interface description language to be exported to the runtime architecture. After serialization, the designer can render a prototype to adjust any undesirable aspects of the interface. Lastly, the final user interface can be deployed on the target platform.

An XML-based meta-language, DynaMOL, is used for the exportation and transportation of models to the runtime architecture. A preliminary implementation of the runtime architecture has been tested on top of the Dygimes framework (Dynamically Generating Interfaces for Mobile Computing Devices and Embedded System) [6]. The architecture consists of a dialog controller which takes care of the changes to user interface. These changes can be caused by user interaction or context changes.

SITUATIONS

Situations[7] is an extension to the context-aware infrastructure called the Context toolkit [8]. It supports easier building of context-aware applications by facilitating access to application states and behavior. It exposes an API to the internal logic of a context-aware application. Context information from sensors or users is made available to the application and application logic is used to acquire and analyze inputs, issue or execute context outputs when appropriate. The application logic consists of creating a “situation” with an information description that it is interested in and the actions associated with the “situation”. It would consist of a number of context rules and the situation will handle the

remaining logic discovery, individual sources of context and data and determining when input is relevant and executing the appropriate services.

Context input is handled by widgets and context output by services. The capture of context input is made easier by the fully declarative mechanism provided by Situations' references. Situations' listeners provide all the necessary functionalities to obtain real-time execution. A default listener called Tracer provides indications of the current status of variables, origins of the variables, and current state of the context rule. Traditional context-aware applications would need to implement customized methods to access application logic. Situations provide standard access and allow arbitrary applications to provide intelligibility and control interaction for context-aware applications and interfaces can be built independent of the main application.

MIMIC

Eisenstein et al. [10] proposed a set of model-based techniques that may aid designers to build UIs across several platforms, while respecting the unique constraints posed by each platform. The approach will isolate the features that are common to the various contexts of use and specify how the user interface should adjust when the context changes.

Knowledge bases are created that describe various components of the user interface, including the presentation, the platform, the task structure, and the context. The knowledge base can then be used to automatically produce user interfaces matching the requirements of each context of use. The user interface is defined as a implementation-neutral description by the MIMIC modeling language. MIMIC is a formal declarative modeling language that comprises of 3 components – platform model, presentation model, and task model.

The platform model describes the computer systems running the user interface and the platform's constraint information. The platform model can then be used to generate a set of user interfaces for each platform. It can also be dynamic and changes accordingly to context changes. The presentation model

describes the visual appearance of the user interface. It describes the hierarchy of windows and their widgets, stylistic choices and the selection and placement of these widgets. The task model represents the structure of the tasks that the user may be performing. It is hierarchically decomposed into subtasks and information regarding goals, preconditions and post conditions may be supplied.

The connections, especially those between the platform and presentation models, are critical to the determination of the interface's interactive behavior. These connections also describe how the various platform constraints will influence the visual appearance of the user interface. Several techniques were described for the creation of connections between the various models and the interpretations of these. However, the automated generation of task-optimized presentation structures using MIMIC has not been developed yet.

REVIEW SUMMARY

Table 1 summarizes the characteristics of the systems reviewed. From the reviews, it can be seen that the common design approach is based on capturing or abstracting contextual information via models or "situations" and then translating these to a required platform. Two main techniques were used predominantly for context modeling – model-based and rule-based. A certain amount of expertise may be found in some of these approaches via the production, storage and maintenance of case-bases, context rules. But these approaches focus mainly on the use of context information for interface generation. None of the systems incorporates dynamic learning capability. Only Gaze-X has a feedback module which enables the user to manually change context actions. In this paper, rule-based technique is chosen for the specification of contextual information while dynamic learning capability is implemented with neural network techniques.

Table 1: Comparison Summary of Surveyed Context-Aware Systems

System	**Development Language**	**Platforms**	**Context Modelling**	**Learning capability**
Supple	Java	Desktop	Model-based	
Gaze-X		Desktop	Context-based	Feedback system
Dynamo-AID	Java, .Net	Desktop, Mobile	Model-based	
Situations	Java, .Net, Flash	Desktop	Rule-based	Runtime change to context rules and variables
MIMIC		Desktop, Mobile	Model-based	
SmartIFace	JME	Desktop, Mobile	Dynamic Context-based	Neural net

SYSTEM DESIGN

The design of the system architecture and learning process are detailed in this section. We first present the problem formulation.

PROBLEM FORMULATION

We define the following for a mobile phone such that $W_T = W_D \cup W_N$:

W_T : set of all widgets for the mobile phone

W_D : set of all widgets displayed on the interface

W_N : set of all widgets not displayed on the interface

Let *C(u)* represent the context tuple for user *u* and *P(w,C(u))* be the probability of selecting widget *w* based on the context *C(u)*. In our *SmartIFace* prototype, we define:

C(u) = [*time*, *location*, *traffic*, *heart rate*, *widget_use*[*day*]]

Then, if *k* widgets are to be displayed such that $W_D = \{w_1, w_2, ..., w_k\}$, we require $P(w_i, C(u)) > P(w_j, C(u))$ for all $1 \le i \le k$ and $w_j \in W_N$.

SYSTEM ARCHITECTURE

The system architecture shown in Figure 1 comprises of the following modules – *GUI Manager*, *Learning Engine*, *RMS* (Record Management System) and *Rule-Base Engine.*

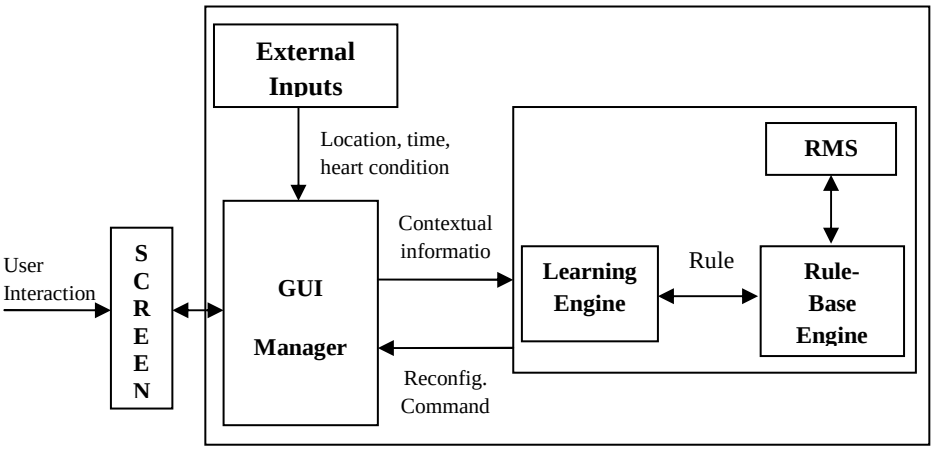


Figure 1: System Architecture

The mobile phone user interacts with the screen and that is usually captured by the phone OS. The proposed system can be viewed to be interposed between the screen and the phone OS. *External Inputs* simulates the supply of sensor- and location-based information to the system. A timer module (not shown) simulates the transition of time for performance analysis and evaluation. Simulated external inputs include GPS location information, time of day, traffic conditions and heart condition. The GUI manager records user interaction when application widgets are accessed. Together, these form the contextual information that is passed to the learning engine. Interface re-configuration is

based on commands from the learning engine referencing rules in the rule-base engine. The RMS handles rule storage in the mobile phone.

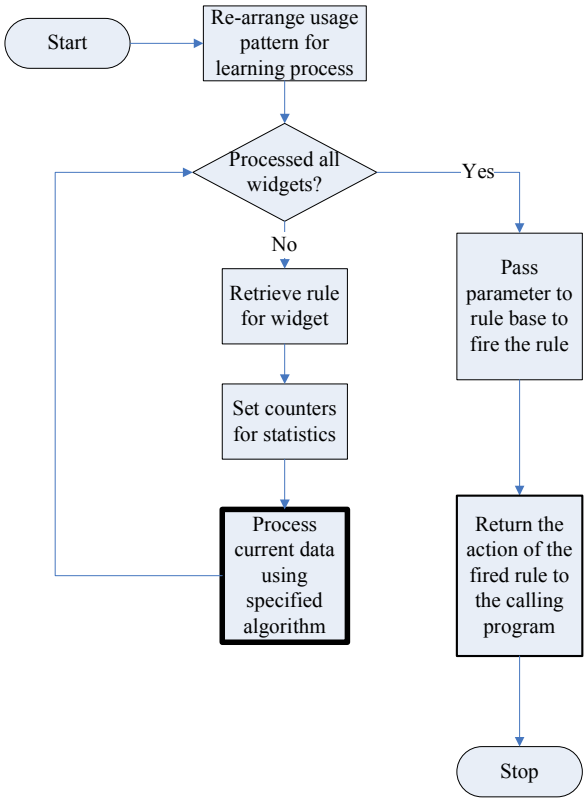


Figure 2: Learning Process Flow

LEARNING PROCESS

The learning engine will add widgets, remove widgets or maintain current screen state based on the results of its learning algorithm. The learning engine communicates with the rule-base engine for rules update and reference. The rule-base engine accepts parameters from the learning engine, fires the appropriate rules and returns the resultant action from the fired rule. The learning process flow is illustrated in Figure 2.

The learning engine is an integral part of the simulation process. The simulation process flow is illustrated in Figure 3. Besides simulating time progression, it will continuously capture users' widget interactions and pass this information to the learning engine for processing. Changes to the context will cause context rules to be updated as shown in the rule-base engine.

The learning engine learns during an initial period of *k* days. In our simulation, we set *k* to 7 for a learning period of 1 week. After this initial period, the rule-base engine continuously communicates with the learning engine at a preset timing. As this may cause some latency in the mobile device's operation, the preset timing was set to midnight when, it is assumed, that user interaction activity would be at its minimum. The timing can, however, be set to any appropriate user-specified timing.

After the learning engine has completed pattern processing and returned the results, a decision will be made on whether to re-configure the screen widgets or maintain current display status. The action taken by the learning engine is determined by the type of learning algorithm implemented (explained in the next section). After the learning engine has performed its action, all widgets for the specified context will be processed for display state changes before the rule-base fires the appropriate rule and returns the action associated with the rule. The rule-base includes helper methods to support rule storage management.

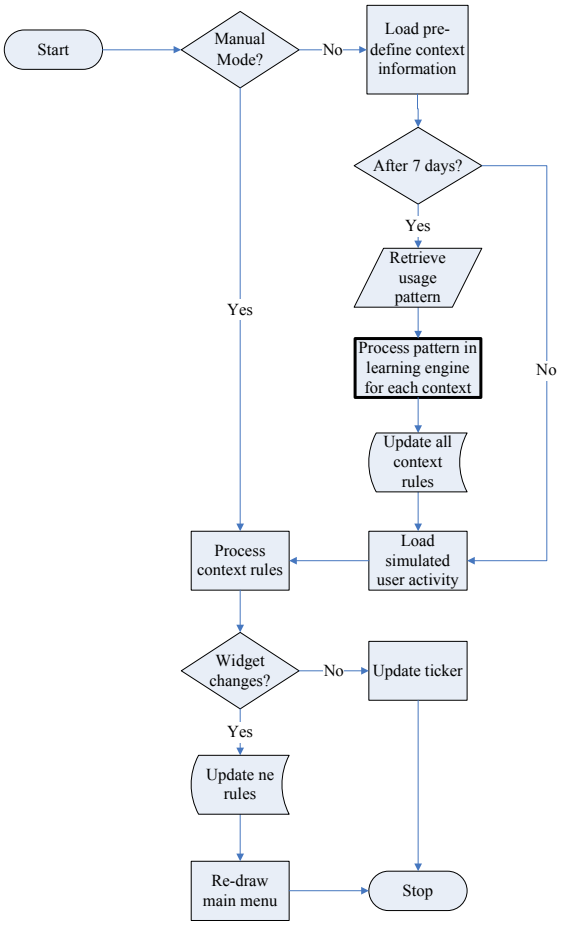


Figure 3: Simulation Process Flow

LEARNING ENGINE DESIGN

The objective of the learning engine is to determine trends from the usage pattern data in the current context. Three different learning algorithms were developed for the learning engine: Minimal Intelligence (MI), Single Layer Perceptron with Error Correction (SLP) and Multi Layer Perceptron with Back Propagation (MLP).

Witmate [30] and Joone [27] make use of several libraries not supported by Java ME, such as the Math class (no support for logarithmic, exponential, power etc), file input/output (text file not supported for Java ME), and event handling. As Witmate is a commercial program, no source code is available. Joone, on the other hand, is open source and may be used for the creation of neural networks. Joone codes, however, cannot be pre-verified by the Java ME platform. Therefore, to ensure complete compliance with the pre-verification process, customized code was developed.

MINIMAL INTELLIGENCE

To reduce processing overheads, the MI algorithm uses only the most recent user activity for all widgets in the phone to decide whether to update rules as shown in Figure 4.

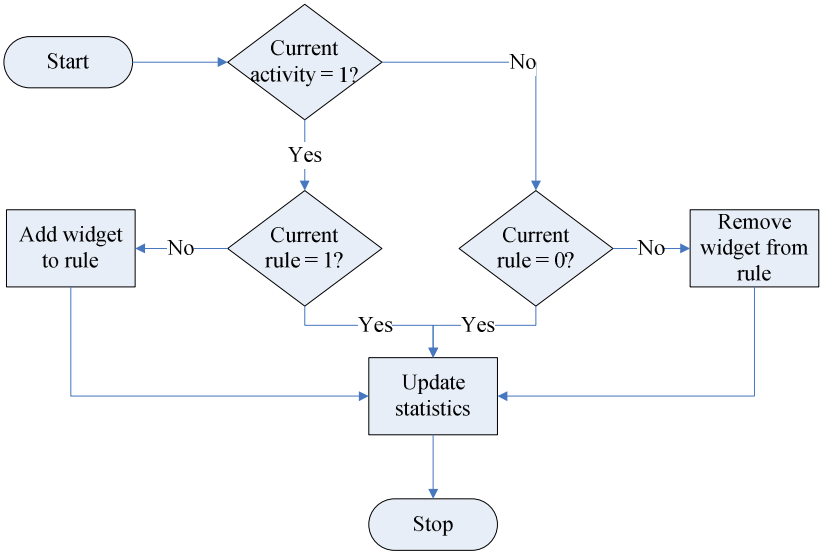


Figure 4: Minimal Intelligence Algorithm Process Flow

Each widget has an indicator in the rule and the widget will be displayed for the context if the indicator is 1. The algorithm first checks if the current user activity for each widget is present or absent (1 or 0). If user activity is present and the rule indicator is 0 (user accessed the widget but widget is not displayed),

it will include the widget in the rule. However, if user activity is absent and the rule indicator is 1 (user did not access the widget but widget is displayed), the widget is removed from the rule. MI does not track user activity pattern over time, only using the most recent user activity data on all widgets to set the rules.

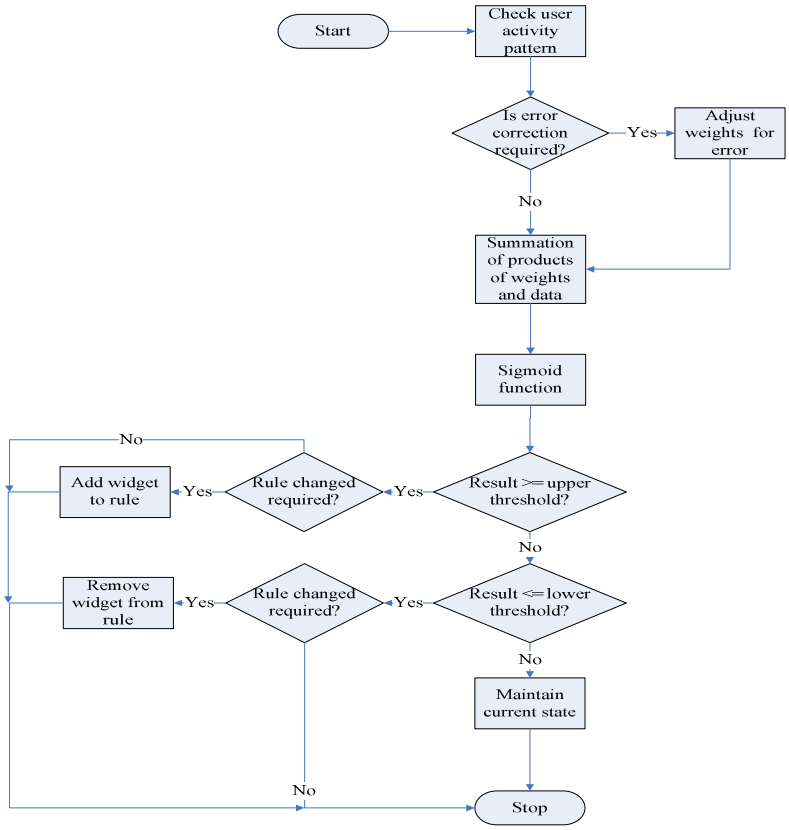


Figure 5: Single Layer Perceptron with Error Correction Process Flow

SINGLE LAYER PERCEPTRON WITH ERROR CORRECTION

The second algorithm is a modified Single Layer Perceptron (SLP) with error correction. The SLP neural network is a simple feed forward neural network that consists of a single processing unit (cell). Each input to the cell is associated with a weight that can be positive or negative to indicate reinforcement or inhibition on the cell. The sigmoid function of the cell sums the products of the

weighted inputs and adds a bias. The bias adjustment (error correction) is based on the previous prediction. The SLP process flow is shown in Figure 5.

MULTI LAYER PERCEPTRON WITH BACK PROPAGATION

The MLP consists of multiple layers of cells and permit more complex, non-linear relationships of input data to output results. There is an input layer, a hidden layer and an output layer. The input layer represents the weighted inputs to the hidden layer. The hidden layer results are computed and used as weighted inputs to the output layer. The output layer uses these weighted inputs to compute the final output. With Back Propagation, the output error is corrected by back propagating this error through the network and adjusting weights in each layer. Convergence can take some time depending on the allowable error in the output. The steps for the back propagation are (for learning data E and expected output C):

1. Compute the forward propagation of E through the network (compute the weighted sums of the network, S, and the inputs, u, of every cell).

2. From the output, make a backward pass through the intermediate layers, computing the error values.
    a. For output cells o : $error_o = (C_o - u_o)u_o(1 - u_o)$
    b. For all hidden cells i: $error_i = (\sum w_{m,i}\ error_o)u_i\ (1 – u_i)$
       m – all cells connected to hidden cell
    c. w – given weight, u – cell input
3. Lastly, update the weights within the network as follows :
    a. For weights connecting hidden to output layers: $w = w + \rho * error_o * u_o$
    b. For weights connecting hidden to input layers: $w = w + \rho * error * u_i$

The forward pass through the network computes the cell inputs and an output. The backward pass computes the gradient and the weights are then updated so that the error is minimized. The learning rate, ρ, minimizes the

amount of change that may take place for the weights. Although it may take longer for a smaller learning rate to converge, it can minimize the chance of overshooting the target. If the learning rate is set too high, the network may not converge at all. The process flow of MLP is shown in Figure 6.

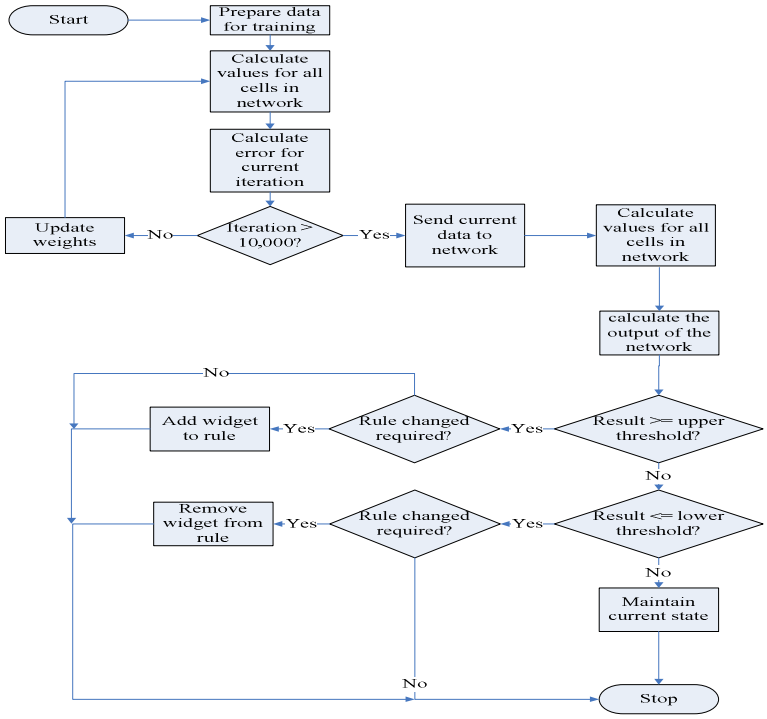
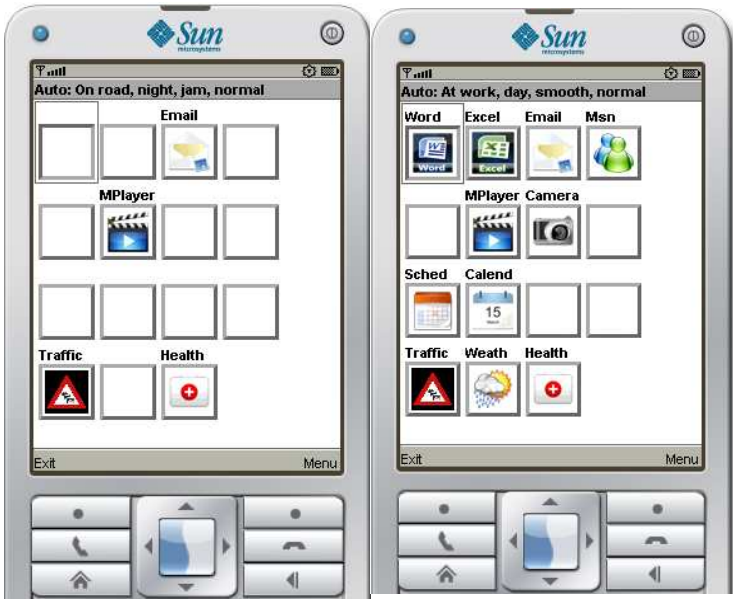


Figure 6: Multi Layer Perceptron with Back Propagation Process Flow

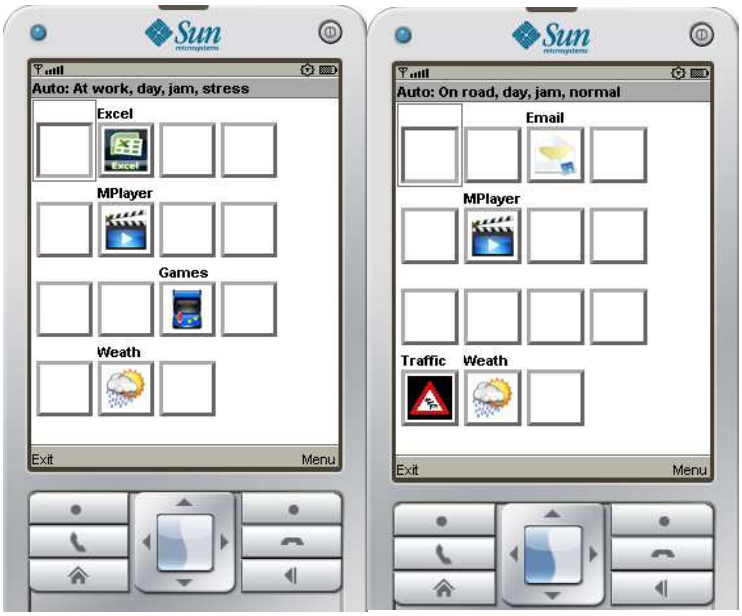
DETERMINATION OF SUITABLE THRESHOLDS

SLP and MLP algorithms require setting appropriate thresholds to determine the output of the neural network. Selecting an appropriate threshold improves prediction accuracy of user activity. However, this is complicated as the output of the sigmoid activation function used is non-linear. Hence, the upper and lower thresholds were designed to be adaptive according to usage data patterns specified for learning.

a b



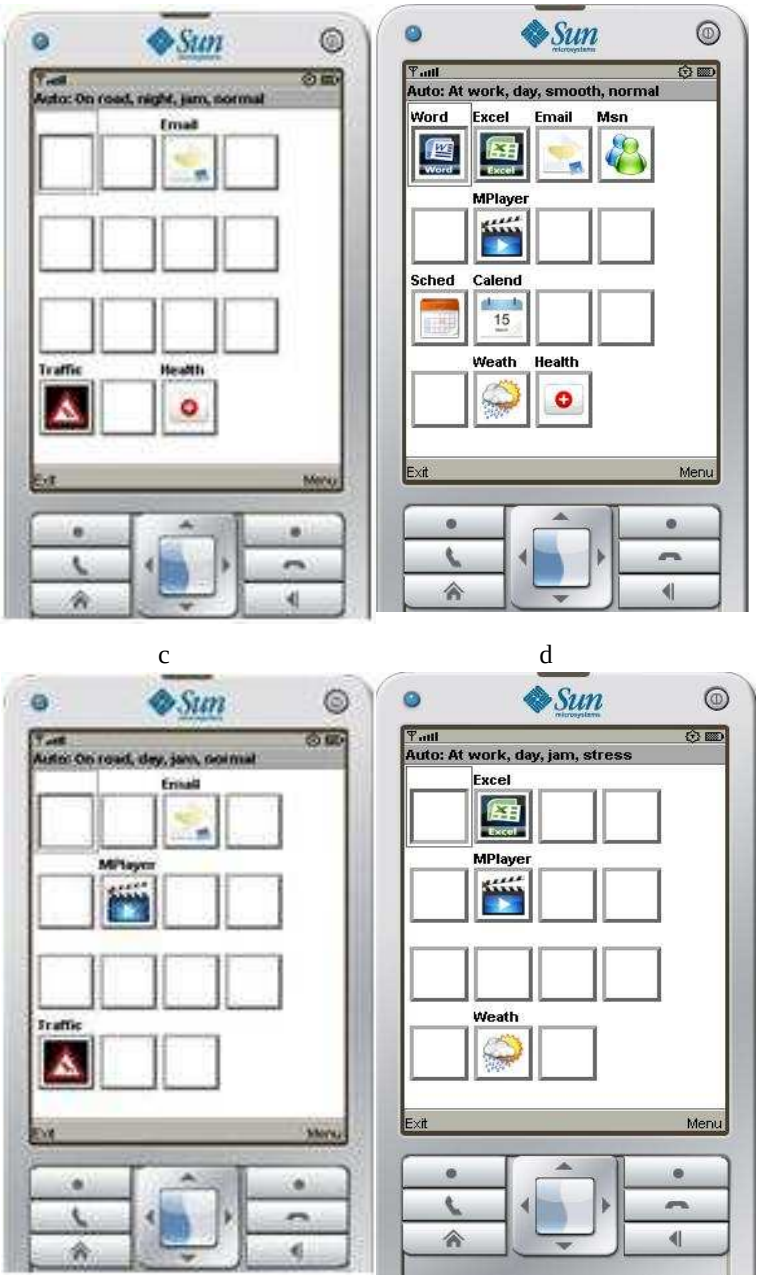
c d



(a) SLP Weekly (Car) (b) SLP Weekly (Office)

(c) MI Weekly (Office) (d) MLP Weekly (Car)

a b



c d

(a) SLP Daily (Car) (b) SLP Daily (Office)

(c) MI Daily (Office) (d) MLP Daily (Car)

Figure 7: Screen Shots of SLP, MI and MLP in action

SELECTION OF TEST SCENARIOS

Since a mobile phone platform is not suitable for file operations, the test environment could not be based on file inputs. Manual data entry was not practical as there was more than 20 days of user activity data. A special function, *loadActivity( )*, was implemented for the loading of test environments onto the mobile phone. Two main categories of test scenarios were created for each learning algorithm – weekly repeating user activity pattern and daily repeating user activity pattern. This enables algorithmic performance under a range of conditions to be analyzed and evaluated. A parameter decides the category of testing scenario to be used and loaded into a vector. At the end of each day, the next day's data is loaded. Figure 7 shows some screen shots in different contexts.

PERFORMANCE TESTING

Testing focuses on the prediction accuracy of usage pattern by the MI, SLP and MLP learning algorithms. Usage pattern comprises widget interaction activity, user location, time, traffic conditions and heart condition. Table 2 summarizes the overall performance.

Table 2: Summary Overview of Prediction Accuracy

	Daily repeating usage pattern	**Weekly repeating usage pattern**
Minimal Intelligence (MI)	57.6%	32.4%
Single Layer Perceptron with Error Correction (SLP)	97%	61%
Multi Layer Perceptron with BackPropagation (MLP)	97%	61.9%

MINIMAL INTELLIGENCE ALGORITHM

Test results show that MI is unable to accurately predict user activities. Prediction is based on the previous day data and the percentage of correct predictions is the lowest among all three algorithms. For the weekly repeating usage patterns (Figure 8), there are not many repeated activity patterns for a given context. This causes the MI algorithm to have low accuracy that tapers off at around 30% after 3 weeks. It is still able to achieve 30% because at any time, four different contexts are used for the learning process and this enables the algorithm to at least distinguish and identify to a certain extent, the context used.

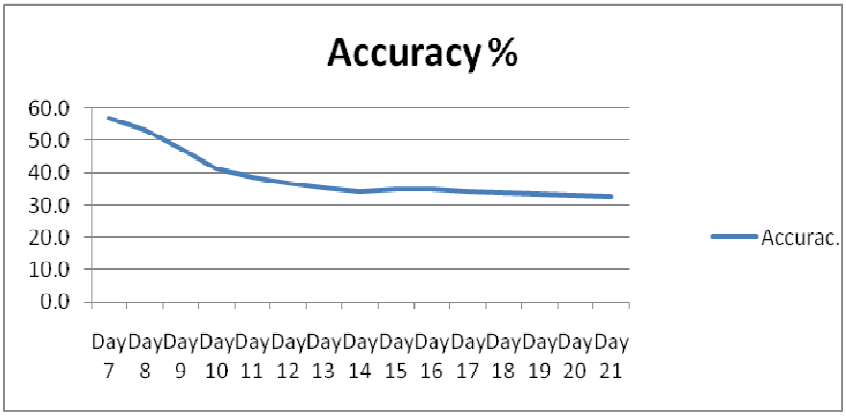


Figure 8: MI Prediction Accuracy for weekly repeating usage pattern

For the daily repeating usage pattern, the MI algorithm is able to achieve higher accuracy than the previous two usage patterns as there are more repeated activities (Figure 9).

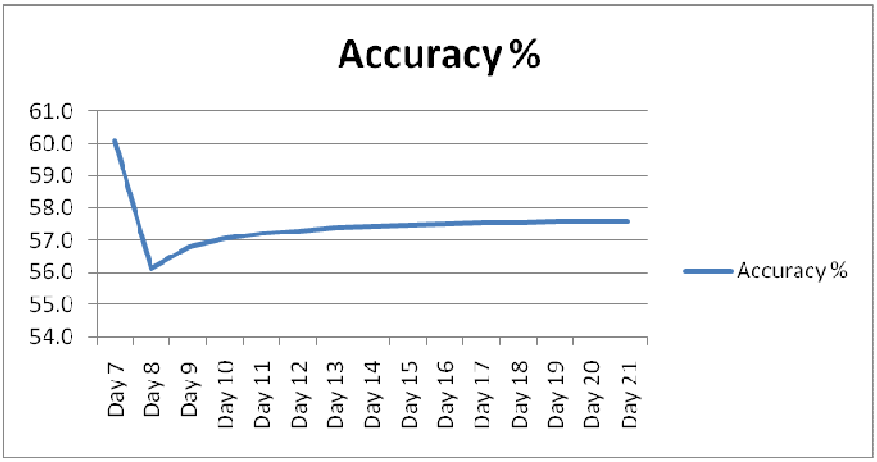


Figure 9: MI Prediction Accuracy for daily repeating usage pattern

SINGLE LAYER PERCEPTRON WITH ERROR CORRECTION

Test results for SLP show that it has the best overall performance among the three learning algorithms. SLP analyzes data preceding the prediction and does its calculations based on different weighted inputs. It does not try to recognize patterns in the input data in order to predict user activity. Instead, weights are re-adjusted based on the next day's data. This removes any problem caused by conflicting input data which can cause inaccuracies in the bias. Results in Figures 10 and 11 indicate that it is better equipped than MI and MLP to predict user activity for both weekly and daily repeating datasets.

For weekly repeating data, SLP gets about 60% prediction accuracy compared to an approximately similar average for MLP (Figure 12). There is no significant difference over MLP. This shows the weakness of the SLP algorithm as it neither learns with the data nor attempts to recognize any data patterns. For highly regular usage patterns, it achieved over 90% prediction accuracy. Usage pattern consistency as with the set of daily repeating data offers inherent data stability to enhance its prediction accuracy.

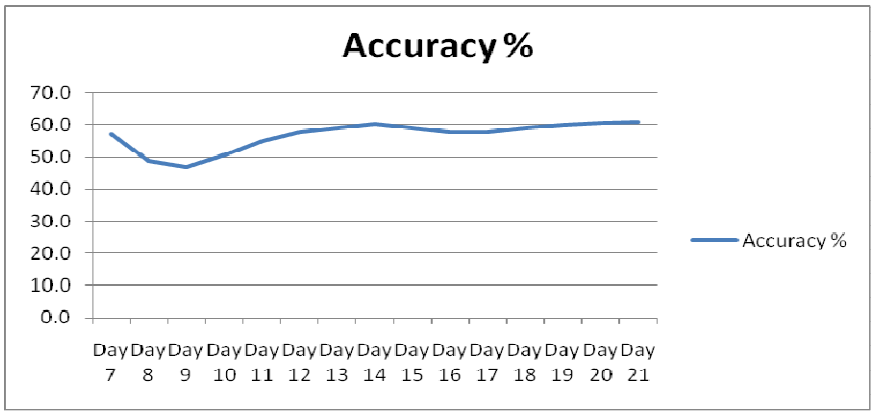


Figure 10: SLP Prediction Accuracy for weekly repeating usage pattern

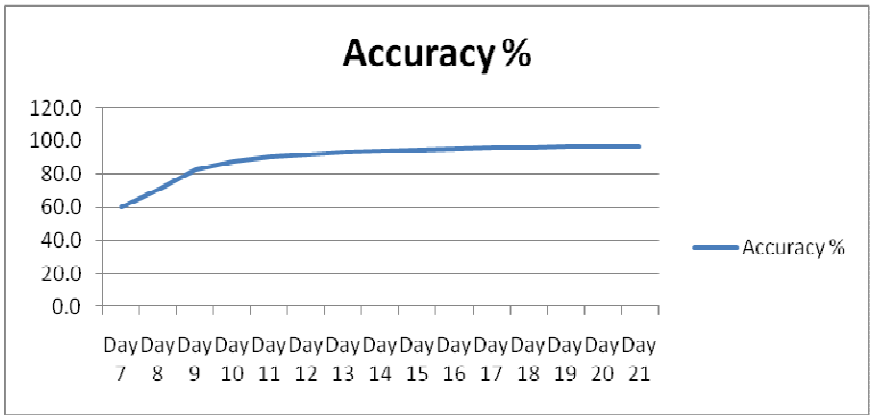


Figure 11: SLP Prediction Accuracy for daily repeating usage pattern

MULTI LAYER PERCEPTRON WITH BACK PROPAGATION

Test results show that MLP has the similar performance to SLP when the usage pattern is regular as with the daily repeating data set (Figure 13). For weekly usage patterns, however, its performance generally trails SLP although the average is similar (Figure 10 and Figure 12). The main reason for this result is because MLP needs to learn from existing data. When the data does not exhibit a significant level of repeating usage patterns, conflicting trends may arise and cause learning errors.

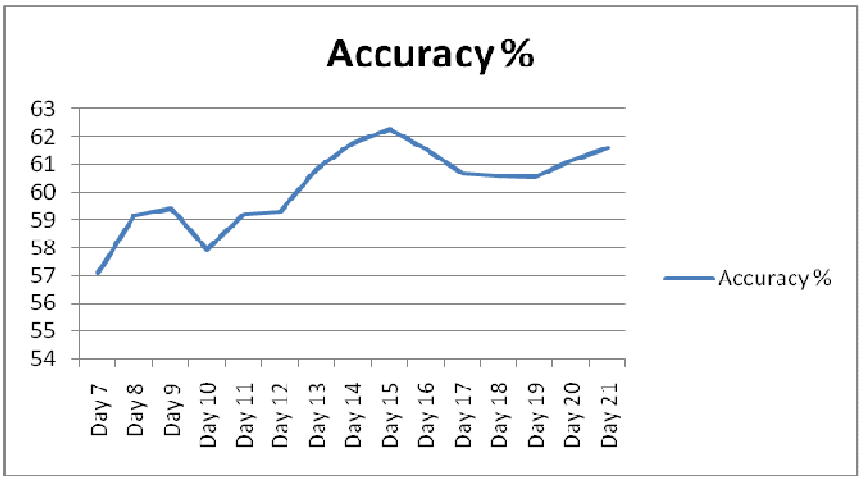


Figure 12: MLP Prediction Accuracy for weekly repeating usage pattern

There are a number of error correction algorithms that can be used with the MLP. These algorithms include Back Propagation, Delta rule and Perceptron. Alsmadi et al. [1] have examined the Back Propagation, Delta rule and Perceptron algorithms and found that Back Propagation gave the best result with the MLP as it is designed to reduce the error between the actual output and the desired output in a gradient descent manner.

Beside the error correction, there are other parameters that may affect the performance of the MLP. The number of hidden layers used and the number of hidden neurons in the hidden layers will in some ways affect the performance of the neural network and the accuracy of the results. Much research has been done on this area but so far there has been no single solution to all problems on deciding the best selection of the parameters. Bishop [2] states that an MLP with one hidden layer is sufficient to approximate any mapping to arbitrary accuracy as long as there are sufficiently large numbers of hidden neurons. However, there is an optimal number of hidden neurons to be used for different networks. Currently, there is no fixed answer to the optimal number of hidden layers and hidden neurons to be used. When there are significant processing steps to be operated on the inputs before obtaining the outputs, then there may be a benefit to having multiple hidden layers. Zurade [20] stated that the number of hidden

neurons depended on the dimension *n* of the input vector and on the number of *M* separable disjoint regions in the *n*-dimension Euclidean input space. He stated that there is a relationship between *M*, *n* and *J* (number of hidden neurons) such that

$$M(J, n) = \sum_{k=0}^{n} \binom{J}{k}$$

It was proved in [19] that the maximum number of *n* nodes is closely related to the *N* training pairs and *d* input dimension in the following formula:

$$n = C \left(N / [(d \log N)] \right)^{\frac{1}{2}},$$

where *C* is a constant

The approach is to try an increasing sequence of C to obtain different numbers of hidden nodes, train the neural network for each *n*, and observe the *n* which generates the smallest root mean squared error. Haykin [12] stated that the optimal number of hidden neurons is a number that would yield a performance near to the Bayesian classifier. His tests showed that a MLP neural network using two hidden neurons is already reasonably close to the Bayesian performance (for his test problem). There are also some rule-of-thumb methods specified in [1] for determining the number of hidden neurons.

In our research, we also performed tests to see if there is an improvement in the performance with different number of hidden neurons. Table 3 shows the result of using different numbers of hidden neurons and their respective prediction accuracies. It is apparent that there is no significant performance improvement observed when using more hidden neurons. Each hidden neuron added to the hidden layer also introduced more lag into the system as more time is required to calculate the output.

Table 3: Summary Overview of Prediction Accuracy

	Two hidden neurons	Three hidden neurons	Five hidden neurons	Tenhidden neurons
Prediction accuracy for weekly repeating usage pattern	62%	62.5%	62.2%	62.1%

With different number of hidden neurons, it was observed that there is an average of 5% increment in the processing time required with each new hidden neuron added. This is especially an important consideration as the processing power available on the mobile platform is very limited. If there is not a big improvement in the performance for using large numbers of neurons and layers, then it would be better to use the minimum required.

For the daily repeating usage pattern, MLP's performance is similar to the SLP algorithm in that it is able to achieve over 90% accuracy due to consistency in the input data patterns (Figure 13). This consistency in the usage data also enables better training of the nueral network. However, the MLP algorithm is observed to introduce a considerable amount of lag into the application due to this training.

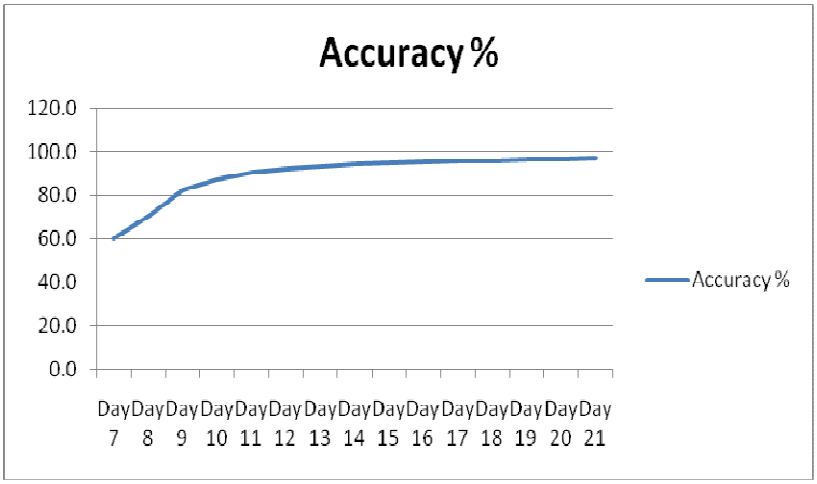


Figure 13: MLP Prediction Accuracy for daily repeating usage pattern

The error correction for MLP is based on mean-squared error reduction (number of iterations required to achieve the acceptable output). To achieve good mean-squared error reduction, the number of iterations must be about 10,000. During testing with the 15 widgets, an average lag of about 200ms was incurred for every learning period. This lag may become significant if more widgets and contexts are involved since the learning duration is proportional to the product of the number of widgets and number of contexts.

SUMMARY

Table 4 summarizes the performance analysis and evaluation of the 3 learning algorithms.

Table 4: Summary of Performance Analysis and Evaluation

Learning Algorithm	Prediction accuracy for regular usage pattern	Processing power requirements	Speed	Complexity
MI	Low	Low	Fast	Low
SLP	Good	Low	Fast	Average
MLP	Good	High	Slow	High

CONCLUSION

In this paper, we have presented the design and development of an intelligent interface reconfiguration engine that is context-aware. Widget reconfiguration is done dynamically without the need for modeling effort. Test results show that both the Single Layer Perceptron with Error Correction and Multi Layer Perceptron with Back Propagation can be used for context-aware reconfiguration of the mobile phone interface. However, the Single Layer Perceptron with Error Correction offers a practical yet effective solution for a resource-constrained mobile phone. It offers low computational overheads with reasonable prediction accuracy for the typical mobile phone user. Although

competitive performance is offered by the MLP, a period of learning with existing data is required. Together with higher computational overheads, it may not be suitable as an on-the-fly approach. Future work would include investigating the effectiveness of approaches that include fuzzy logic engines and/or the Kohonen neural network as well as deploying the system on an actual mobile phone integrated to suitable wireless sensor device inputs.

REFERENCES

1. Alsmadi, M. K. S., Omar, K. B., & Noah, S. A. (2009). Back Propagation Algorithm: The Best Algorithm Among the Multi-Layer Perceptron Algorithm. *International Journal of Computer Science and Network Security*, 9(4), 378-383.

2. Bishop C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, ISBN: 0198538642, 116-160.

3. Bontempi, G., Birattari M., & Bersini,H. (2002). New learning paradigms in soft computing. *Physica-Verlag Studies In Fuzziness And Soft Computing Series*, 97-136.

4. Clerckx, T. Luyten, K. & Coninx, K. (2004). DynaMo-AID: A Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. *Engineering Human Computer Interaction and Interactive Systems, Lecture Notes in Computer Science*, 3425, (pg 871-876). Springer.

5. Clerckx, T., Winters, F. and Coninx, K. (2005). Tool Support for Designing Context-Sensitive User Interface using a Model-Based Approach. *Proceedings of the 4th International Workshop on Task Models and Diagrams for user interface design*, (pg 11-18). ACM Press.

6. Coninx, K., Luyten, K., Vandervelpen, C., Bergh, J. V. D., & Creemers, B. (2003). Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. *5th International Symposium*

in Human-Computer Interaction with Mobile Devices and Services, Lecture Notes in Computer Science, 2795, (pg 257-272). Springer.

7. Dey, A. K. & Newberger (2009). A. Support for Context-Aware Intelligibility and Control. *Proceedings of the 27th international conference on Human factors in computing systems*, April 2009, (pg 859-868). ACM Press.

8. A. K. Dey, Gregory D. Abowd,and Daniel Salber (2001). A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction Journal*, 16(2-4), 97-166.

9. Du, W. & Wang L. (2008). Context-Aware Application Programming for Mobile Device, *Proceedings of the Canadian Conference on Computer Science and Software Engineering*, Vol. 290, (pg 215-227).

10. Eisenstein, J., Vanderdonckt, J., & Puerta, A. (2001). Applying Model-Based Techniques to the Development of UIs for Mobile Computers. *Proceedings of the 6th international conference on Intelligent user interfaces*, (pg 69-76). ACM Press.

11. Gajos, K. Z. & Weld, D. S. (2004). SUPPLE : Automatically Generating Personalized User Interfaces. *Proceedings of the 9th international conference on Intelligent user interface*, (pg 93-100). ACM Press.

12. Haykin, S. S. (1999). *Neural networks: a comprehensive foundation.* 2nd Edition, Prentice Hall.

13. Heaton, J. (2008). *Introduction to Neural Networks with Java*. 2nd Edition, Heaton Research.

14. Henricksen, K. and Indulska, J. A Software Engineering Framework for Context-Aware Pervasive Computing. *Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications*, (pg 77-86). IEEE Computer Society.

15. Kurniawan, S., Mahmud, M. and Nugroho, Y. (2006). A Study of the Use of Mobile Phones by Older Persons. Proceedings of the *Conference on Human Factors in Computing Systems*. (pg 989-994). ACM New York.

16. Maat L. and Pantic M. (2007), Gaze-X : Adaptive Affective Multimodal Interface for Single-User Office Scenarios. *Proceedings of the ICMI 2006 and IJCAI 2007 international conference on Artifical intelligence for human computing*, (pg 171-178). Springer-Verlag.

17. Satyanarayan, M. (1996). Fundamental Challenges in Mobile Computing, *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. (pp 1-7). ACM New York.

18. Schmidt, A. (2006). Implicit Human Computer Interaction Through Context. *Personal and Ubiquitous Computing*, 4(2-3), 191-199.

19. Xu S. & Chen, L. (2008). A Novel Approach for Determining the Optimal Number of Hidden Layer Neurons for FNN's and Its Application in Data Mining. *Proceedings of the 5th International Conference on Information Technology and Application.* (pg 683-686). Springer-Verlag.

20. Zurade, J. M. (1992). *Introduction to Artificial Neural Systems*. PWS Publishing Company.

21. Android SDK. Retrieved Feb 2011, from http://developer.android.com/index.html

22. BBC News. Retrieved Feb 2011, from http://news.bbc.co.uk/2/hi/technology/7833944.stm

23. HTC Hero. Retrieved Feb 2011, from http://www.mobilitysite.com/2009/07/htc-hero-widgets/

24. HTC Hero Scenes. Retrieved Feb 2011, from http://www.gsmarena.com/htc_hero-review-382p3.php

25. HTC Touch Pro 2. Retrieved Feb 2011, from http://pockethacks.com/htc-touch-pro2-proximity-sensor-demo
26. iPhone SDK. Retrieved Feb 2011, from http://developer.apple.com/iphone/
27. Joone. Retrieved Feb 2011, from http://sourceforge.net/projects/joone/
28. Nokia 5500 tilt sensor. Retrieved Feb 2011, from http://tech2.in.com/india/reviews/smart-mobile/nokia-5500-sport/3824/1
29. S310 motion sensor. Retrieved Feb 2011, from http://www.mobilefanatic.net/2006/06/samsung-motion-sensor.html
30. Samsung Blade S5600v. Retrieved Feb 2011, from http://www.dintz.com/review-samsung-blade-gt-s5600v/
31. Witmate available. Retrieved Feb 2011, from http://www.witmate.com/