Route Prediction from Cellular Data

Kari Laasonen

Basic Research Unit, Helsinki Institute for Information Technology
Department of Computer Science, University of Helsinki
Kari.Laasonen@cs.Helsinki.FI

Abstract. Location-awareness is an important concept in pervasive computing. Using programmable mobile phones we can equip users with software that tracks cell transitions in a GSM network. With this location data and other context variables we can determine places that are important to the user, and make predictions about the next location when the user is moving. The predictions and location data can be made available to others, in a form of a presence service. The learning algorithms work solely on the user's personal phone and do not need any external server infrastructure. Because the user is in control of the movement data, we avoid otherwise problematic privacy issues. This paper describes the task of learning routes and predicting future locations by maintaining a database of physical routes. Route predictions are based on approximate string matching techniques. When a confident prediction cannot be made, we attempt to predict at least the general direction of movement by finding places where different routes fork.

1 Introduction

Location awareness plays a large role in ubiquitous computing. Several applications have been proposed that rely on knowing or predicting the location of the user. Not merely reacting to a known location, but trying to accurately predict human movement is a very ambitious research problem.

We present an algorithm for predicting movement from cell-based location data. Such location data consists of transitions between cells, with no regard to physical locations or topology. Given this apparently scant data, we are still able to learn, on user's personal mobile device, places that are personally important to that user, and to make predictions about the place the user is moving to. Such predictions are useful in, e.g., a presence service, which makes the whereabouts of the user available to other people. Many other proactive applications also become possible if we know the semantic associations of a location or a future location.

Existing approaches to learning important locations and predicting routes often rely on coordinate data such as GPS [4,2]. However, GPS can be problematic in urban areas due to signal shadowing. Another issue is that of privacy. By storing location information on the user's own phone, we avoid the inherent privacy issues of some external agent constantly collecting and processing our whereabouts.

This paper works with the conceptual model presented in Laasonen et al. [3]. The contribution of the present paper is an enhanced algorithm for predicting routes. The algorithm analyzes whole paths using string processing techniques, instead of relying on the short path fragments of the earlier paper. This method both conserves memory and offers better prediction accuracy.

Section 2 describes the prediction problem and the general approach in more detail. The actual route prediction algorithm is presented in Section 3. The performance of the method is evaluated in Section 4, which is followed by concluding remarks.

2 Problem Description

2.1 Locations and Bases

A GSM phone communicates over the air with a base station. In any given location there may be several base stations whose radio signal reaches the phone. The phone chooses one of them, and switches transparently over to a new base station as needed. A *cell* is the area covered by a single base station; when we say the phone is in some cell, we mean that the phone is in the area of the corresponding base station.

In addition to overlapping each other, there are several properties to cells that makes for challenging data analysis. Cells vary widely in size, and signal shadowing can make cells appear non-contiguous. Finally, a certain physical location does not have one-to-one correspondence to cells because of radio interference, phone network load and various other issues. On the other hand, GSM phones are ubiquitous and cellular networks are present almost everywhere. Since no operators or external service infrastructure are involved, data gathering is easy and inexpensive.

The software records each *cell transition.* At the lowest level each cell is represented by an opaque numeric identifier (e.g., "*Sonera.3286.15754*"), so our location data is a time-stamped sequence of such identifiers. We can visualize the data by making a graph where the vertices are the observed cells, and there is an edge (c_i, c_j) if (and only if) a transition occurred from c_i to c_j. A fragment of such graph is shown in Fig. 1. This graph shows both the author's daily commute from home ("Vuosaari") to work and trips from home to downtown Helsinki. It does not include transitions in the opposite direction. The names in this graph were supplied manually, although some cellular operators have now begun to offer location services.

If overlapping cells have approximately equal signal strength, the phone may hop between cells even when the user is not moving. We handle this oscillation by *clustering* cells. Intuitively, a cell cluster is a group of nearby cells where most transitions happen within the cluster. For a precise definition of cell clusters and an algorithm for forming them on-line, see [3].

A *location* is either a cell cluster or a single cell. Locations are identifiable in the sense that we can reliably detect the user entering and leaving them.

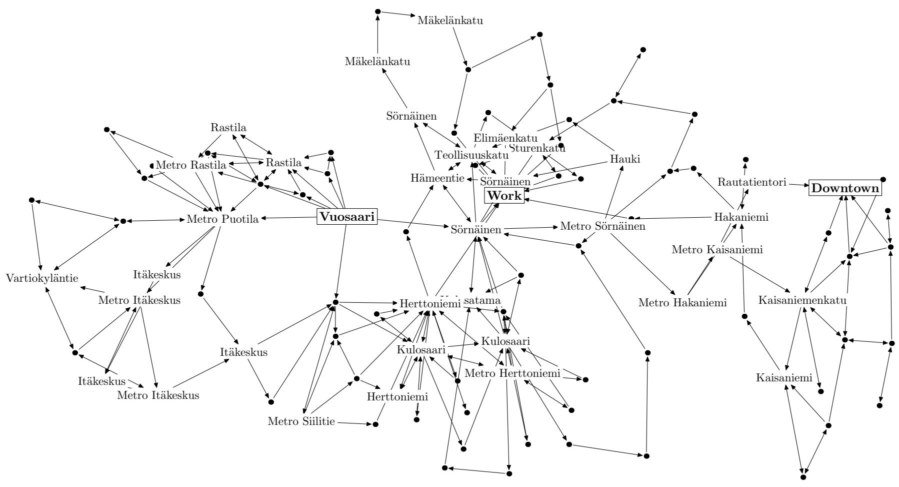

Fig. 1. A partial cell transition graph for routes from "Vuosaari" to either "Work" or "Downtown." Unlabeled dots represent cells that have not been named. The data comes from 69 separate trips.

Finally, locations that are important to the user are known as *bases*. A location is promoted to a base when the time spent there as a portion of the total time the software has been run goes above a certain threshold. The set of bases can change over time, as new places become important or old ones are not visited as often. The problem of determining bases is covered in [3]; we work with a set of known bases.

2.2 Route Prediction

Perhaps the the most important consequence of using cell-based location data is that we lack the physical topology of the cell network. This includes the correspondence between cells and physical locations, and also all indications of direction. Seeing cell sequence ABA could mean that the user visited B and came back. Or then cell A was just briefly shadowed by B. Looking only at the immediate context is all but useless. We have make inferences from larger base of context information.

There are two basic approaches to the problem. The first is to examine the local context of recent cells [3]. Suppose we are in cell c and the previous cells have been $h_1, h_2, \ldots$. The idea is to prepare strings $h_k h_{k-1} \ldots h_1 c$, for varying values of k. These strings are matched against a database of previously stored fragments. Based on the matches found, and the bases reached from c, we get probabilities for the next base. We begin with, say, $k = 4$. For larger values of k we are wasting storage space, because most longer sequences are seen rarely. With smaller k we will get more matches, but at the same time lose the essential context information that we need to establish direction. This method can be augmented with the use of time distribution, which can help us to distinguish between similar routes.

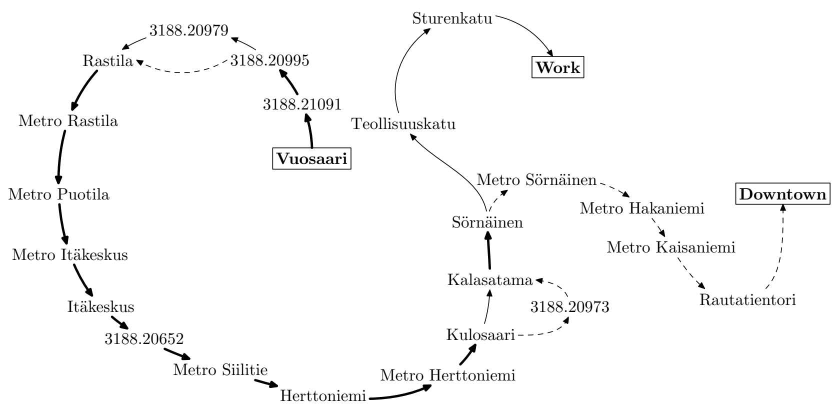

Fig. 2. The most frequent composite routes from "Vuosaari" to "Work" (thin line) or to "Downtown" (dashed line). Edges appearing on both routes are shown with a heavy line. Unnamed cells have numeric identifiers only.

The second approach, and the topic of this paper, is more global. Instead of using the local context, we look at entire routes between two bases. We attempt to learn all different physical routes as strings of cell identifiers. Whenever the user completes a route r between bases a and b, we determine if an existing route between a and b is similar to r. If such a route is found, the two routes are merged together. To make a prediction, we proceed as follows. Since we know the user has left base a, we have a set of possible routes and their destinations b. We now use a recent history h of cells and find the route that exhibits the largest similarity to h.

Figure 2 shows the effect of applying route clustering to the data of Fig. 1. There are five different physical routes; the two most frequently traveled are shown in the figure. The graph is obviously vastly simpler, and furthermore corresponds quite closely to the routes actually traveled in the physical world.

Treating routes as a whole enables a number of features not possible with the fragment method. First, we can detect *fork points*. A fork point is a place where overlapping paths diverge, such as "Sörnäinen" in Fig. 2. When there are several good similarity matches, we can offer a fork prediction as an insurance against the actual base prediction going amiss. From the point of a presence service, a high-confidence prediction of the fork is probably more useful than several low-confidence base predictions.

We can also detect *backtracking*, which happens when the user physically goes some place that is not a base, and comes back. By looking at the entire path, it is not difficult to see if its suffix resembles some earlier subpath, but in reverse. Finally, *loop* routes, which return to the base from which they started, are rather common. An interesting future research problem is to reliably detect the turning point on such routes. A much more useful prediction would then result from subdividing the route at the turning point and adjusting predictions accordingly.

3 Prediction Algorithm

The problem of route prediction is to predict the next actual base b^*, given that the user's last base was a and since then we have seen a cell sequence $c_1, \ldots, c_k$. There are three phases in the algorithm. First, each time the user leaves a base, that is, enters a cell c not part of the current base, the system prepares for a new route prediction task. At each cell transition, we make a prediction, which is a set of pairs (b, p), where b is a possible future base and p the probability of the user going there. Finally, when the user arrives at a base, the entire route $a, c_1, \ldots, c_n, b^*$ is used to make better subsequent predictions.

3.1 Route Composition

For each pair of bases a and b we maintain a set of routes R_{ab}. When making predictions, we need to match the cell history against the routes R_{ab} for all possible b. Instead of storing every encountered path in full, we aim to keep only "typical" paths. Not only does this decrease the memory requirements substantially, but it also proves crucial in estimating the relevance of a given route.

A new route $t = ac_1 \ldots c_n b$ is added to the database when the user arrives at base b. Suppose now that the maximum similarity of t against all $r \in R_{ab}$ occurs with some $r = r_{\max}$ and is greater than a threshold value σ. Then t is merged with route $r_{\max}$. If, however, the similarity falls below σ for all existing routes, we add t to R_{ab}, the set of (distinct) routes between a and b. This process resembles incremental clustering, where each route acts as a cluster, attracting similar routes. The similarity function $\mathrm{sim}(r, t)$ is described below in section 3.2.

The merging of two paths tries to produce a composite path that retains the features of both (similar) participants. We treat the paths as simple strings of cell identifiers, or just "letters." First the two path strings are aligned by adding empty elements ("spaces") to both. As far as possible, identical letters will appear in the same position in both aligned strings. Computing an alignment of strings s_1 and s_2 is similar to finding their edit distance, which gives the number of editing operations needed to transform s_1 into s_2. Both of these can be computed with a well-known dynamic programming algorithm [1].

Next we give each letter in both strings a position (or value) in the range $[0, 1]$. If the string is $x_1 \ldots x_n$, the initial value assignment to ith letter is simply $v(x_i) = (i - 1)/(n - 1)$. Now the merged position of letter x is the average position of all nearby occurrences of x in both strings. This averaging is only performed within a small window in order to handle cyclic paths (which begin and end at the same location) correctly. The merged string results from arranging the letters in ascending order by merged position. If two or more letters have the same merged position, their ordering is undefined. In subsequent merging letters with undefined positions receive identical position values.

For example, suppose we are merging strings "`twirls`" and "`tries`". The optimal alignment is

```
t w i r l ␣ s
t r i ␣ ␣ e s
```

The value v of the first letters is 0, the value of the second letters ('w' and 'r') is 1/6, and so on. Computing the average value we find that

$$\begin{aligned} v(\mathtt{t}) &= 0, & v(\mathtt{w}) &= \tfrac{1}{6}, & v(\mathtt{i}) &= v(\mathtt{r}) = \tfrac{1}{3}, \\ v(\mathtt{l}) &= \tfrac{2}{3}, & v(\mathtt{e}) &= \tfrac{5}{6}, & v(\mathtt{s}) &= 1. \end{aligned}$$

The merged string is thus "tw(ir)les", where the parentheses indicate that 'i' and 'r' share the same position. It has turned out important to know that some cells do not necessarily have fixed order to them.

The procedure described so far produces a complete merging of the two routes. A *composite route* is a merged route which does not include cells seen significantly fewer times than the average. When the user leaves base a, we obtain the possible destinations $S = \{b_1, \dots\}$ and the composite routes to all of these destinations.

3.2 Route Similarity

The similarity function $\mathrm{sim}(r, t)$ between two routes is used by the prediction algorithm in two cases. In both cases r is a composite route between two bases. The other parameter t can be a complete path, as above, when the similarity determines the clustering of routes; alternatively, it can be a history of cells (about 10 most recently seen cells), and we want to find the route that most closely resembles the history.

To estimate the similarity of event sequences, Moen [5] describes a scheme which uses the edit distance coupled with event-level similarity. Events are considered similar if they appear in similar contexts. Substitution among similar events carries very small cost. The problem with this approach is that event similarity matrix takes quadratic time and space; typical phone memories would fill up in a few months of operation.

To find a simpler heuristic method, we start from the Jaccard measure $J = n_{rt}/(n_r + n_t - n_{rt})$, where n_r and n_t are the number of elements in r and t, respectively, and n_{rt} is the number that is in both. The measure J is symmetric, but unfortunately ignores direction, so a string is equivalent to its reverse.

An *inclusion similarity* I is similar to J, but asymmetric: strings r and t are considered equivalent if every element in t appears in r in the same order. This asymmetry derives from the fact that a composite route typically contains more cells than there are in any actual instance of that route. We thus let $I(r, t) = T/|t|$, where T is the number of elements in t that are found, in-order, in r. For example, $I(\mathtt{abcdef}, \mathtt{acbdg}) = 3/5$; letters 'a' and 'c' are in order, but 'b' and 'c' have been exchanged. The letter 'd' is again in proper order with respect to 'c'. In cyclic paths there is at least one cell x which appears in r more than once. For any such x we choose the instance that yields the largest similarity for the entire string. Experimentation shows that the function I fulfills the desired property of yielding results that are virtually indistinguishable from those produced by the much more involved event sequence methods.

3.3 Making Predictions

We are now ready to make a prediction based on the previous base a and the already seen cell path $c_1 c_2 \dots c_k$. The entire path is used only to detect backtracking; for route matching, a history $h = c_{k-m} \dots c_k$ of the most recent is used instead. The reason not to use the entire path is twofold: we can detect faster that the user is stepping outside any known path, and using shorter strings is in general more efficient.

Route matching has produced a set S of possible reachable bases when starting from base a. Making a prediction entails computing for each candidate base $b \in S$ the similarity

$$\sigma_b = \max\bigl\{ \operatorname{sim}(r, h) \bigm| r \in R_{ab} \bigr\}.$$

In other words, σ_b is the largest similarity of the cell history against all routes leading to b. A very simple prediction system would stop here, and predict that the next base is the b that maximizes σ_b. If we recognize we are on certain path, it is logical to surmise that this path will be followed to its destination. However, several routes can have nearly equal similarities. We can choose between by using additional context variables. Another option is to instead find a fork point, as discussed in the next section.

The context variables that are present in the data are time of day, weekday and cell frequency. The basic idea is that we can store for each encountered cell its frequency and time distribution. The previous base a is used as additional context, so that the *context database* C appears as a set of associations $(a, c) \to \bigl(b, n, T(c)\bigr)$. Such entries mean that at cell c, having started from base a, we have ended up n times at b. We use $T(c)$ to denote the time distribution parameters at location c. Although this scheme is already quite conservative in its consumption of memory, it is possible to still reduce the memory usage by ignoring all intermediate cells. Intuitively, if we know the distribution of times at the previous base, knowledge of these at each c does not offer much additional information. In this reduced model we have a database C' where the associations take the form $a \to \bigl(b, n, T(a)\bigr)$. In both cases the context database is updated when we arrive at a new base.

From the context database we get another set of bases, defined as $R = \bigl\{ b \bigm| \langle (a, c) \to (b, \dots) \rangle \in C \bigr\}$ for the full database C (the case C' is defined analogously). Now the probability of going to base b when located at c at time t can be written as

$$\begin{aligned} p_b = P(b \mid a, c, t, D) &\propto P(b, t \mid a, c, D) = P(t \mid a, b, c, D) P(b \mid a, c, D) \\ &\propto n P(t \mid a, b, c, D), \end{aligned}$$

where D represents all the data we have seen on previous trips. The remaining task is to find the probability of being on the given route at time t. It is assumed that time of day t_d follows normal distribution, so we need to store in $T(c)$ the sum s and the square sum q of the previous event times. Then t_d is normally distributed with mean $\mu = s/n$ and variance $\sigma^2 = q/n - \mu^2$. For the weekday t_w the normality assumption works less well, so the frequency is used instead.

If t_d and t_w were assumed to be independent, we could simply write $p_b \propto nP(t_d)P(t_w)$. We can compute $P(t_d)$ directly from the normal distribution function, normalizing the probabilities in the end. In reality, this assumption does not hold; instead, we should compute $P(t_w \cap t_d) = P(t_w)P(t_d|t_w)$. This can be handled by maintaining a separate normal distribution for each weekday. Since this increases the memory requirements substantially, it was implemented only for the initial-time context C'.

The process just described yields a probability p_b for all bases $b \in R$. To complete the prediction is to examine bases $b \in R \cap S$. Each such b gets a weight $\sigma_b p_b$, where σ_b is the route similarity computed for all $b \in S$; the base with the largest weight will be our prediction. If $S = \varnothing$ or contains nothing but very low-similarity bases, only R is used; conversely, if R is empty, we try again to construct R with the second or third most recent cell in history. Only if this process fails a few times do we resort solely to bases in S.

3.4 Finding Fork Points

Finding fork points depends on the assumption that each possible candidate path corresponds to a different physical route. Unless the path merging is done carefully, we can end up with composite routes with bits and pieces from many separate trips. Let $P = \{r_1, r_2, \dots\}$ be the set of candidate paths, where each r_i begins at the current cell and thus includes only possible future cells. Now the *fork cell* f must be some cell that appears on all the routes; otherwise the user could choose some path r_j with $f \notin r_j$. Then find a cell sequence $f_1, \dots, f_n$ such that every $f_i \in \bigcap_j r_j$ and let $f = f_n$. In other words, the fork point is the last cell that appears on every candidate path. (The ordering can be computed by taking from each r_i only the cells in the intersection and merging the resulting paths, as described in section 3.1.)

4 Evaluation

The algorithms were evaluated on dataset presented in [3]. The data was collected for six months in 2003 with an early version of the ContextPhone software [6] running on a Nokia 7650 phone. The movements of the three volunteer users were tracked both at work and at leisure. The movement patterns range from very simple (daily commute, some weekend and holiday trips) to moderately complex.

The baseline algorithm is the fragment-based method [3], which was tested with several window sizes k. Since the algorithms are intended for small devices, their memory consumption is also investigated. To simplify the evaluation, both algorithms were tested with offline bases, that is, the bases are given at the beginning of the simulation.

The algorithms were implemented as simulations that received cell transition events one at a time and were requested to give a prediction for the next base. The resulting prediction was then compared to the actual base, which was known

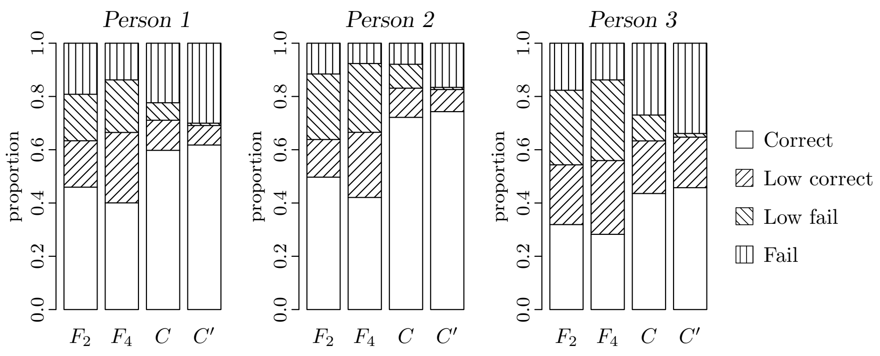

Fig. 3. Route recognition accuracy for various prediction methods.

only to the driver code. Again following [3, sect 4.4], we exclude cases when the user is apparently not moving (stationary), or when the next base has not been seen earlier.

Figure 3 shows how the different methods compare. There are three graphs, one for each test person. Each graph shows how the various prediction algorithms performed. The F_2 and F_4 are the fragment method with a window size of 2 and 4, respectively. The symbol C denotes the route prediction using the normal context database, which maintains a time distribution for all intermediate route cells; the reduced model C' has a time distribution only for the starting times.

A prediction is *correct* if it matches the actual next base and the probability of the given prediction is larger than $u = 0.3$. A *low correct* prediction is one that is correct, but probability is less than u, or the second-best prediction is correct with nearly equal probability (e.g., $p_1 = 0.55$ and $p_2 = 0.44$), or the fork point was predicted correctly. A *low fail* prediction was wrong, but the probability was also low. Finally, a *fail*-type prediction was a high-confidence prediction that went wrong, or no prediction at all.

For all persons, the route-based method supplied more predictions that succeeded. Taking into account also the low-confidence correct predictions, however, we see more moderate improvements. The conclusion is that the route-based method is an improvement on the fragment method when it comes to prediction accuracy. However, it is not known which level of accuracy is attainable for an automatic prediction mechanism. Almost certainly this level varies with different people.

It is interesting to note that although models C and C' are very similar in their prediction accuracy, the latter uses much less memory, as shown in Fig. 4. But even model C consumes less memory than any fragment-based method. For the latter, memory use consists of the fragments themselves and the associated storage for context predictors. The route-based method needs less predictor memory, preferring compact route descriptions. The number of separate routes varies widely, from 107 of Person 2 to 1254 of Person 3. (The dataset contained 414 and 4290 trips between two bases, respectively.)

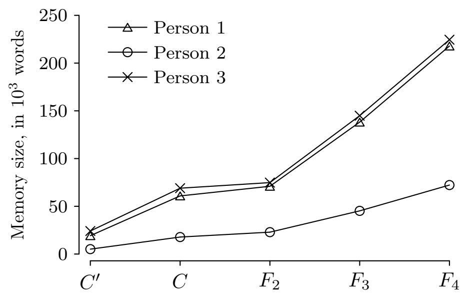

Fig. 4. Comparison of the memory consumption of the algorithms.

Because the proposed algorithm is a combination of two separate predictors, it is fairly oblivious to parameter changes. Setting the similarity threshold σ to a low value uses less memory, because more routes will considered equivalent. This weakens the prediction based on route similarity, but the change is offset by the time-based predictor. The same effect occurs by setting the history length m to a very low value. The tests were run with $\sigma = 0.7$ and $m = 12$, which provide a good compromise between quality and efficiency.

5 Conclusion

We have presented a method for predicting user movement from cellular data gathered with user's own mobile phone. The algorithm tackles the problem by attempting to recognize physical routes traveled by the user. Later predictions are based on matching the current cell history against known routes. The current time is used to aid prediction.

The method is an improvement over the previous one both in prediction quality and memory usage. It works very well for people having fairly simple patterns of movement, but the results are not optimal for more people traveling a lot. The learning model is still very simple; better methods are needed to decide when time-of-day inferences help and when they do not.

References

1. Gusfield, D.: *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press (1997)
2. Harrington, A., Cahill, V.: Route Profiling—Putting Context to Work. In: *2004 ACM Symposium on Applied Computing SAC'04*, ACM Press(2004) 1567–1573
3. Laasonen, K., Raento, M., Toivonen, H.: Adaptive On-device Location Recognition. In *Pervasive Computing: Second International Conference*, LNCS 3001, Springer Verlag (2004), 287–304
4. Marmasse, N., Schmandt, C.: A User-centered Location Model. *Personal and Ubiquitous Computing* **6** (2002) 318–321
5. Moen, P.: Attribute, Event Sequence, and Event Type Similarity Notions for Data Mining. Ph.D. thesis, *Report A-2000-1*, University of Helsinki (2000)
6. Raento, M., Oulasvirta, A., Petit, R., Toivonen, H.: ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications. *IEEE Pervasive Computing* **4** (2005) 51–59