LocaRhythms:
Real-Time Data Mining for Continuous Detection and Prediction of Stays

Mirko Fetter, Tom Gross
Faculty of Media
Bauhaus-University Weimar
99423 Weimar, Germany
<firstname.lastname>(at)medien.uni-weimar.de

***Abstract*—In distributed teams information on each other's whereabouts is an important prerequisite for efficient communication, coordination, and cooperation. In this paper we present LocaRhythms—a novel approach for continuous detection and prediction of users' stays. This is achieved by continuous capturing of position data, by real-time data mining with an innovative combination of machine learning algorithms, and by anytime presenting stays and follow-up stays.**

***Keywords*—Computer-Supported Cooperative Work; Mobile Computing; Real-Time Data Mining; Continuous Detection and Prediction.**

I. INTRODUCTION

The members of distributed teams need information on each other's presence and availability as an important prerequisite for efficient communication, coordination, and cooperation [10]. For instance, the users' whereabouts are an essential part of this information, and can help other users to decide where somebody should be contacted now or later.

The users' whereabouts can be entered manually (e.g., in many social software systems such as FaceBook [6] and Twitter [28] users frequently make manual entries). Alternatively, the whereabouts can be detected automatically (e.g., via GPS, or GSM). These approaches have the disadvantage that they either entail effort for the users (if they have to do it manually), or that they are of limited value to users (if they only present the GPS coordinates with latitude and longitude in a very specific point in time).

In this paper we present LocaRhythms—a novel approach for continuous detection and prediction of users' stays. The up-to-the-moment models of this approach can generate valuable information for users and their proximity to each other. For instance, if user A is moving to user B's location, user B can get an automatic request to wait. LocaRhythms can, at any given time, produce the accurate stay of a user and the most probable next stay the user will move to. This is achieved by continuous capturing of position data, by real-time data mining with an innovative combination of machine learning algorithms, and by anytime presenting stays and follow-up stays.

The paper is structured as follows. First we introduce the LocaRhythms concept consisting of continuous capturing, real-time data mining, and anytime presenting. Then we present the LocaRhythms implementation with the GPSCoordsSensLogger, the LocaRhythmsServer, the LocaRhythmsClient. Finally, we discuss related work and draw conclusions.

II. LOCARHYTHMS CONCEPT

The following requirements for the LocaRhythms concept are user-driven (see [11] for a good overview paper). Basically a mobile device with positioning capabilities should be used to get information on users' whereabouts. By means of machine learning techniques a user's whereabouts should be detectable and predictable for any given time, based on the respective user's movement patterns. This information should be useable, for instance, for arranging meetings, for estimating each other's availability, or for enriching the users' feeling of connectedness over distance.

For describing the LocaRhythms concepts we define the following terms, based on Merriam-Webster's dictionary [16], with some adaptations: *space* is 'a boundless three-dimensional extent in which objects and events occur and have relative position and direction', in our notion it exists per se; *location* is 'a position or site occupied or available for occupancy or marked by some distinguishing feature', in our notion it can be seen as a sub-space and exists per se (e.g., a restaurant, a shopping mall); *coordinates* or geographic coordinates or GPS coordinates are the latitude and longitude with an optional altitude of a point in a location in a space, in our notion it does not exist per se but can be measured and detected; *trace* is 'a mark or line left by something that has passed', in our notion it typically consists of a set of coordinates that can be measured and detected and combined; *significant place* is a location, in our notion it is of importance to users expressed by the duration and frequency of past visits; and *position* is a 'point or area occupied by a physical object', in our notion it is the current coordinates of a specific user and can be measured; and *stay* is 'a residence or sojourn in a place', in our notion it consists of a set of the most current positions.

Based on these notions we can now describe the concepts consisting of continuous capturing, real-time data mining, and anytime presenting.

A. *Continuous Capturing*

A first step is to learn what places are of significance to a specific user and how these places can be distinguished from the space that is just transited by the user. A widely used approach for identifying significant places is to record the users' movement with position-tracking devices and to extract the significant places from the collected data. The approaches mainly vary in the way the coordinates are captured and the significant places are inferred [2, 14, 15, 20]. In LocaRhythms we decided to use GPS devices for tracking users' coordinates, since this allows us to find significant places with a fair accuracy in a wide area without the need to setup environments and infrastructures. Also, GPS does not have limitations of other technologies such as: location technologies based on radio frequency that only cover a small area with good accuracy (e.g., those based on the IEEE 802.11, or Bluetooth), or location technologies based on GSM that cover a wide area with an insufficient accuracy (e.g., GSM positioning via the Cell ID). Current GPS devices have additional advantages for our approach: they are small enough to be carried around in the pocket, offer worldwide positioning out doors and are affordable. In some recent mobile phones GPS chips are built in—so, no additional hardware is needed. The most recent generation of GPS chips (e.g., the SIRFstarIII [23]) have the strengths to go through windows and thin walls and can, therefore, be partly used for indoor positioning. So, with the SIRFstarIII used in LocaRhythms data can be captured in our offices and in some private and public locations.

Still, GPS has—despite the fact that we were using SIRFstarIII in LocaRhythms—some limitations in terms of imprecision of the captured data. First, the GPS coordinates and traces recorded indoors have a greater amount of outliers due to reflections of the signal. So, the traces need to be pre-processed and filtered to minimise the number of outliers. Second, GPS is in some conditions prone to errors; for instance, in LocaRhythms sometimes no proper GPS data can be captured, because there is no signal indoor (in which case null-values are retrieved), because the GPS device runs out of battery (in which case no data are retrieved), or because the users forget to turn the tracking on (in which case no data are retrieved). So, the system has to be robust enough to deal with periods were no data can be captured.

B. *Real-Time Data Mining*

The real-time data mining works as follows: first, we detect significant places; and second, we build prediction models that allow statements on the most probable stay of a user and the most probable following stay.

Detecting Significant Places

In order to *detect* and infer the significant places from the traces of the captured GPS coordinates, we tested two clusterers: simple k-means because of its popularity, and expectation-maximisation because of its strengths over simple k-means. Simple k-means is easy, but needs the number of expected clusters (i.e., the number of significant places) as an input parameter, which is not adequate for a flexible system that should learn and grow in the number of significant places identified. Expectation-maximisation allows for a flexible number of clusters. A considerable drawback of both algorithms is that they are not considering the time stamp of the GPS coordinates. Consequently, when applied to a set of coordinates, the importance of places is underestimated compared to transits, which can forge the calculated centroids of the detected significant places—for instance, when the traces of a user often cross at a certain location, this location will also be recognised as a significant place.

We, therefore, decided to omit these standard clustering algorithms and to use a specific algorithm: the time-based clustering algorithm introduced in [13]. The algorithm takes GPS traces and works sequentially through the coordinates contained in order to identify the significance of a place defined by the length of time spent there. If a user stays in a defined radius for longer than a defined threshold for the first time, this location becomes a significant place. When a user stays in a space where significant places have already been identified, the fusion algorithm checks for overlaps. If the new location overlaps with a known location, the algorithm merges the two locations and calculates the centroid of the new merged significant place. If not, a new significant place is created.

In order to better deal with periods in which no proper GPS data can be captured (especially if the signal is lost and when due to this interruption null-values are captured) we extended the algorithm. The extended algorithm stores the last proper position and coordinates its coordinates with the first coordinates that are captured after the interruption and applies heuristics about their relationship.

If the signal is regained near the position it was lost, the algorithm assumes that the person stayed in a covered area (e.g., a building) that shielded the signal and that this is a significant place for the user (e.g., in our test data, this heuristic could identify the university main building, and the shopping mall nearby as two distinct significant places).

The comparison of the two locations needs some tolerance towards deviations. This was for two reasons. Firstly, when users went in a covered area, the signal was lost immediately, but when they left the area it could take some time until the GPS device regained signal (typically some 100 meters away from the exit). Secondly, when users went in a covered area through one entrance, but exited through another (typically from a shopping mall). We built a tolerance heuristic that considers the users' movement through a covered area as well as out of a covered area. For the time no signal is retrieved, it incrementally grows the temporary radius of a significant place with a fraction of the human walking speed until it reaches a defined maximum radius (our practical observations revealed that a maximum radius of 500 meters is best). If the user exits the covered area within the range, then it is considered a significant place (e.g., a shopping mall); otherwise, it is considered a transit (e.g., an urban canyon or a long tunnel with limited or no GPS coverage). With this LocaRhythms tolerance heuristic we were able to detect a considerable number of significant places that would be lost otherwise. So with this setup we were able to get a satisfying set of significant places from the test data of the two authors and of a lab student collected during their everyday routines over different periods.

Predicting Significant Places

In order to predict the place a user will most likely visit at a given time, a statistical model is needed. Hidden Markov Models (HMM) [21] are often used for different modelling and classification problems and proved to work good with time series data. Among the fields were HMMs have been applied successfully are, for instance, speech [12], activity [22] and gesture recognition [29]. Since the basic requirements for predicting significant places are similar, we chose HMM for the prediction of the next most likely place a person visits at a specific time based on the sequences of visits of this person.

In our setting the algorithm needs a user's stays as input. A user's stay is detected when identifying significant places. When a significant place is identified, the algorithm not only stores the significant place, but also captures the time the stay began, and the time the stay ended and stores these times together with the coordinates of the significant place's centroid. Based on theses stays LocaRhythms builds individual Hidden Markov Models for each time span: for each month of a year, for each day of a week, and for each hour of a day. Each of these 43 models is continuously trained with observation sequences. Each time a stay at a significant place is inferred, the time of this stay is split up into time-slices of 100 seconds each. Based on these time-slices, the 43 sets of observation sequences are continuously built and updated depicting the transitions between places for each time-slice. Finally, each set of observation sequence is used to compute the state transition probabilities of an HMM, where the HMM states represent the significant places, and the HMM transition probabilities represent the transition probabilities between places. Based on these models, LocaRhythms makes a range of predictions: using basic weighting algorithms between the individual models, statements on the most probable place at a given hour for a given day as well as the next probable states following this stay can be made.

C. Anytime Presenting

In order to present the results to the users LocaRhythms uses Google Earth [8] for a visual access to the whereabouts of a user's peers. Please note, that in order to protect the users' privacy, users need to explicitly confirm that they want to share the data about their whereabouts. The map-based visual presentation allows users to easily grasp significant places and stays, which would be considerably harder with text-based presentations of the raw GPS coordinates. Overall, LocaRhythms provides four presentation types:

The two basic presentation types of LocaRhythms are the presentations of the ten most important significant places of a user with the respective importance: one based on the number of visits; and the second based on the overall duration of stays. LocaRhythms provides a view in Google Earth where for both presentation types the places are presented by spheres, with their radius representing their respective importance.

The two advanced presentation types of LocaRhythms are the presentations of places a user is most probable at a given time (cf. Figure 1). The LocaRhythms client allows users to specify a time span they want predictions for. With this time span, LocaRhythms can generate two basic queries and send them to the server: The first bar visualisation query with the specified hour and weekday provides the most probable place for this period as a white bar in the map and a set of the next probable states also as bars, depicting their probability by the height of their darker filling colour. The second cylinder visualisation query allows selecting an interval in the form of two weekdays and the start and end hour. The result of this query is a map with cylinder drawn into it at the significant places of the user, subdivided in 24 equal segments. Each segment represents an hour of the day and the darkness of a segment indicates the probability of the user staying at this place at that specific hour.

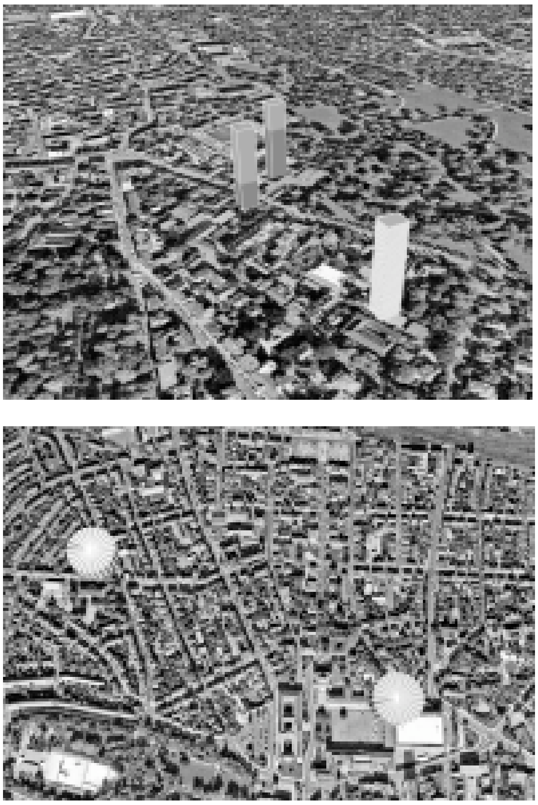


Figure 1. LocaRhythms advanced presentation types with bar visualisation and cylinder visualisation in Google Earth.

III. LOCARHYTHMS IMPLEMENTATION

The LocaRhythms implementation consists of the GPSCoordsSensLogger subsystem for continuous capturing of the GPS traces of a user, the LocaRhythmsServer subsystem for real-time data mining on these traces, and the LocaRhythmsClient subsystem for presenting the significant places (cf. Figure 2). The implementation is based on Java SE 5.0 [27] and J2ME [25] and optimised to run on Mac OS X 10.4.9 on a standard Macintosh PowerPC G4 1.8 Gigahertz.

A. Continuous Capturing in the GPSCoordsSensLogger

In order to facilitate the collection of GPS data the GPSCoordsSensLogger subsystem was implemented. By supporting a variety of devices we make the collection of data as convenient for users as possible. A version for PDAs and mobile phones supporting MIDP 2.0 and CLDC 1.1 was implemented in J2ME (cf. Figure 3) and a second version for laptops was implemented in J2SE (cf. Figure 4). Both versions collect GPS data and save the data in a log file or directly send them to SensBase via XML-RPC [30] if the device is online.

In both versions the GPSLocation component continuously reads the data from the GPS mouse—in our case an Royaltek RBT-2010 with a SIRFstarIII chipset—via Bluetooth using Java's Bluetooth API [24] and parses the NMEA sentences [18] for GPS coordinates in WGS84-Format [17] and hands back a QualifiedCoordinates object to the GPSCoordsSensLogger class in a five second interval. Each QualifiedCoordinates instance holds the latitude, longitude, and altitude (Lat./Lon./Alt.) values as well as the values for horizontal and vertical dilution of precision (HDOP/VDOP) as an indicator for the impreciseness of the measured position based on the satellites' position. The J2ME version is extended with the LAPILocation component for mobile phones supporting the Location API [25] in order to access the built-in GPS capabilities. The LAPILocation class hands back the current position as QualifiedCoordinates to the GPSCoordsSensLogger every five seconds. When online, the GPSSensor component directs each GPS coordinate packed into SensorEvent to the SensBase subsystem of the LocaRhythmsServer via XML-RPC. In both versions the GPSCordsSensGUI component allows the user to monitor the capturing. In J2SE the GUI is based on Swing, in J2ME the GUI is based on the MIDP GUI model.

B. Real-Time Data Mining in the LocaRythmsServer

For the real-time data mining in the LocaRhythmsServer subsystem the SensBase subsystem takes the incoming streams of SensorEvent from the GPSCoordsSensLogger and builds up chains or directed non-cyclic graphs of InferenceEngines (IE) that execute individual logical modules of the overall generation of models. The SensBase subsystem is implemented with the Sens-ation platform [9]. The Sens-ation platform offers mechanisms to easily integrate new IEs and to build up connections between instances of IEs at runtime. In order to facilitate the building of the connections, an instance of the respective IE is registered to listening for incoming events of specified sensors and writes out the inferred results to a new sensor as SensorEvent. The IngredientsID of SensorEvent acts as a pointer to the SensorEvent determining the result. This allows later reproduction of inference chain. In LocaRhythms these mechanisms set up a network of interdependent IEs for analysing incoming position data. Several IEs are part of the LocaRhythmsServer: InferenceEngineSignificantPlaces and InferenceEnginePlaceFusion detect significant places based on time-based clustering; InferenceEngineJAHMMLocation predicts locations

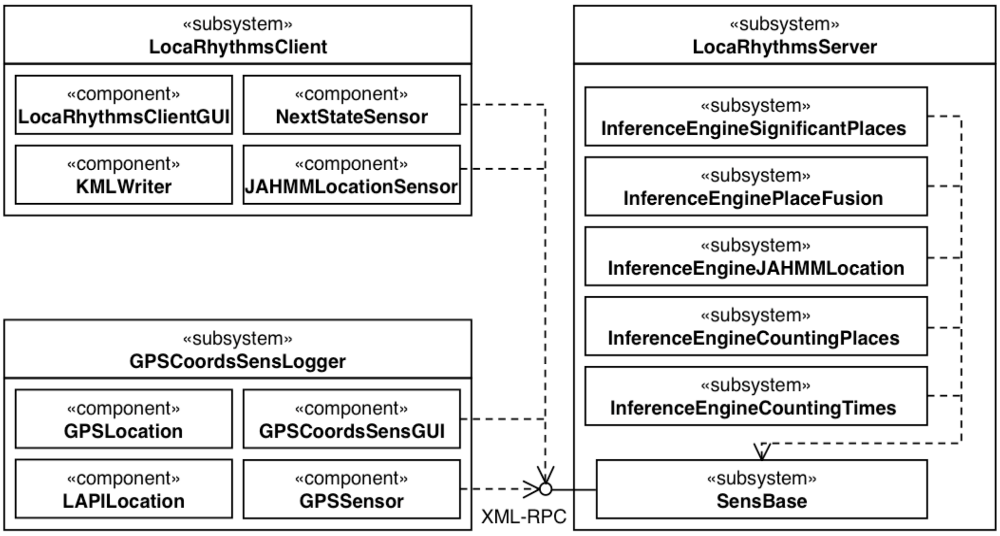
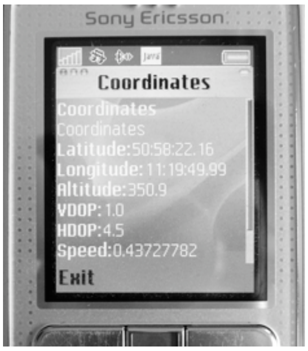


Figure 2. LocaRhythms component diagram.



Coordinates

Coordinates
Coordinates
Latitude:50:58:22.16
Longitude: 11:19:49.99
Altitude:350.9
VDOP: 1.0
HDOP:4.5
Speed:0.43727782
Exit

Figure 3. GPSCoordsSensLogger on a mobile phone.

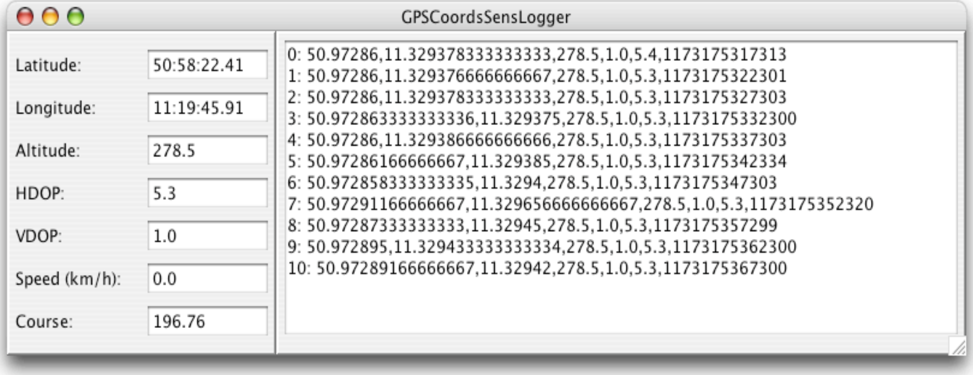
based on HMM; and `InferenceEngineCountingPlaces` as well as `InferenceEngineCountingTimes` aggregates stays according to the number of visits and the overall duration of stay based on calculations.

For all users a graph of IEs is instanced to process their traces. It starts with the `InferenceEngineSignificantPlaces`. The `InferenceEngineSignificantPlaces` is notified by `SensBution`'s `InferenceEngineHandler` every time a new `SensorEvent` containing GPS coordinates is sent to the observed sensor by the `GPSCoordsSensLogger`. As LocaRhythms uses the Weka toolkit for data mining [31] the first step in the IE's `notify()`-method is to transfer each `SensorEvent` into a Weka instance object for further processing. In the next step the incoming coordinates are filtered using the `PRIMIGPSFilter` (an adapted version of Weka's `RemoveWithValue` filter) to remove each instance with a HDOP value greater then 5, as they are considered to be too imprecise. The remaining coordinates are added to the `TimeBasedClusterer` (the LocaRhythms implementation of the time-based clustering algorithms as proposed by Kang et al. [13], as a Weka extension). With a threshold for time t and distance d, the algorithms filters out the places with the radius d a user stays for time t and discards the transits between them. Based on Kang et al. we chose 300 seconds for t, but slightly decreased the value for d to 15 metres based on the better accuracy of our GPS compared to the WiFi positioning of Kang et al.

For the tolerance heuristic the `TimeBasedClusterer` slowly increments the distance d with a fraction (i.e., one fifth) of the walking speed at 1 metre per second resulting in 0.2 metre per second up to a threshold of 500 metres is reached for d. So, for instance, if the GPS signal is retrieved after 15 minutes, d will have grown from its original 15 to 195 metres.

When a new significant place is detected the coordinates of its centroid are written out as a `SensorEvent` to the specified output sensor together with the timestamps when this place was visited and left. This sensor is observed by an instance of the `InferenceEnginePlaceFusion`. This IE checks if a new discovered significant place is within d/3 of a previously detected place. If the distance is smaller the two places are merged and the centroid is updated. This happens every time a user returns to a previously found place. Otherwise a new place is added to the vector of `relevantPlaces` and a unique ID, the `placeID`, is generated for this place. In both cases the updated or new relevant place is written out to the IE's output sensor together with the `IngredientIDs` of each previous `SensorEvent` that are included.

Subsequently the `InferenceEngineJAHMMLocation` is notified by the output of the `InferenceEnginePlaceFusion`. Figure 5 depicts an activity diagram of the `InferenceEngineJAHMMLocation`. The `InferenceEngineJAHMMLocation` builds up a series of HMMs based on JAHMM—a free HMM implementation in Java [7]—and manages them in the class `JAHMMLocationHMMs` that acts as the data store. When a `SensorEvent` arrives the IE retrieves the beginning and end of the last stay via the `IngredientID`. Stays that last over two or more hours are split into smaller portions of one hour each and converted into `JAHMMLocationStays`, consisting of the `placeID` and the beginning and end time of a stay. This information is then added to a `JAHMMLocationModel` for the month, the weekday, and the hour that the stay took place for each `JAHMM-LocationStay`. If the respective model does not exist, it is created beforehand. Furthermore, a `JAHMMLocation-Coordiantes` is added to the model and links the `placeID` to a specific GPS coordinate. If the `placeID` already existed, the coordinates are overwritten with the new coordinates that reflect the



Latitude: 50:58:22.41
Longitude: 11:19:45.91
Altitude: 278.5
HDOP: 5.3
VDOP: 1.0
Speed (km/h): 0.0
Course: 196.76

```
0: 50.97286,11.329378333333333,278.5,1.0,5.4,1173175317313
1: 50.97286,11.329376666666667,278.5,1.0,5.3,1173175322301
2: 50.97286,11.329378333333333,278.5,1.0,5.3,1173175327303
3: 50.972863333333336,11.329375,278.5,1.0,5.3,1173175332300
4: 50.97286,11.329386666666666,278.5,1.0,5.3,1173175337303
5: 50.97286166666667,11.329385,278.5,1.0,5.3,1173175342334
6: 50.972858333333335,11.3294,278.5,1.0,5.3,1173175347303
7: 50.97291166666667,11.329656666666667,278.5,1.0,5.3,1173175352320
8: 50.97287333333333,11.32945,278.5,1.0,5.3,1173175357299
9: 50.972895,11.329433333333334,278.5,1.0,5.3,1173175362300
10: 50.97289166666667,11.32942,278.5,1.0,5.3,1173175367300
```

Figure 4. GPSCoordsSensLogger on a notebook.

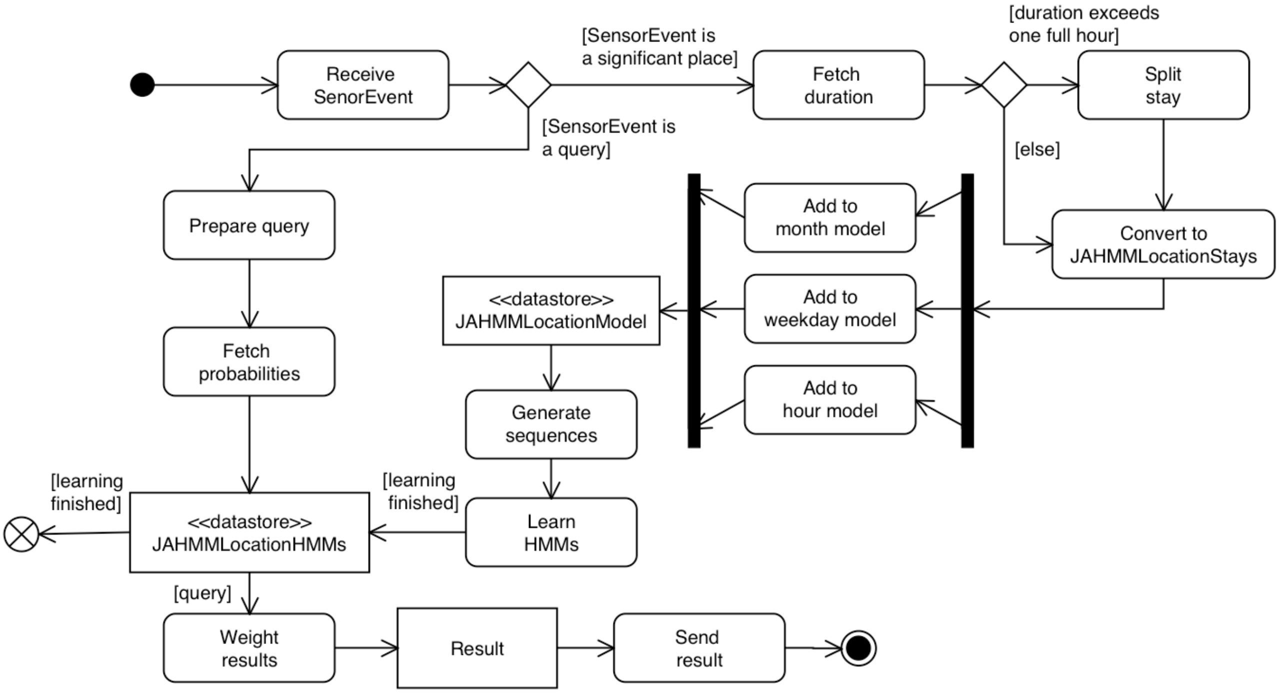


Figure 5. InferenceEngineJAHMMLocation activity diagram.

fine displacement caused by the previous merging of the centroids. The JAHMMLocation-SequenceGenerator now updates the set of observation sequences for all three updated models by splitting up the stays in smaller fractions and generating seq-files that reflect the locations visited in a period by a line of placeIDs separated by semicolons (cf. Figure 6). Finally, a HMM model is generated based on the seq-file by the SequenceToJAHMM-LocationModelConverter using JAHMM. When a query is sent to the IE, the probabilities are fetched from the appropriate models and a simple weighting function combines them to the most probable state as described in more detail in section C.

Furthermore, the output sensor of the Inference-EnginePlaceFusion notifies the InferenceEngine-CountingPlaces and the InferenceEngineCounting-Times for an overall ranking of the significant places. The InferenceEngineCountingPlaces compares the incoming placeID of a significant place with a HashTable of the revisited significant place. If the placeID is in the HashTable, the variable counting the visits to this significant place is incremented. If a new placeID is received, the significant place is added to the HashTable. This way the InferenceEngine-CountingPlaces generates a HashTable on how often each significant place is visited that is continuously updated. In much the same way the InferenceEngine-CountingTimes calculates the overall time a user stays at a significant place. When the IE is notified it can retrieve the beginning and end of the last stay at this place via the IngredientIDs and calculate the duration of the last stay. This duration is added to the overall duration of stays at this place in a HashTable the IE manages. If a user visits this place for the first time, the place and the duration of the last stay are added to this HashTable. Both IEs output an updated, sorted list of places with either the number of visits to this place or respectively the overall duration of stays at this place.

Each of the previously described IEs stores their output into SensBase as SensorEvent. Most of the IEs described here, additionally rely on an internal model that is not only reflected by the input and output values, but computed over time and needs to be available at runtime. For fault tolerance, each IE serialises its internal state to a file. We integrated a mechanism that can—depending on the special needs of each IE—serialise a whole IE to a file. When SensBase is restarted after a downtime, and the IE are re-instantiated, the system restores the last serialised state of each IE from file. Each IE can trigger the serialisation individually in specific time intervals or on basis of incoming or outgoing SensorEvents. This way, the IE can assure that the current state is always available

```
4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;2;2;2;2;2;2;2;2;2;2;2;
4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;4;2;2;2;2;2;2;2;2;2;
4;4;4;4;4;4;4;4;4;4;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;
7;7;7;7;7;7;7;7;7;7;7;7;7;7;7;7;7;7;7;7;2;2;2;2;2;2;2;2;
```

Figure 6. Four observation sequences for the 11 o’clock model representing the times slot of 11 to 12 am on 10, 11, 12, and 13 February 2009.

in event of a system downtime.

C. Presenting Anytime in the LocaRhythmsClient

For the presentation to the user a simplified GUI allows to generate queries to the LocaRhythmsServer. The results of these queries are passed back to the LocaRhythmsClient in form of a SensorEvent. Based on this answer, a file in the Keyhole Markup Language (KML) format [19], is created, that visualise these results. KML is an XML-based language that was developed to define annotations and visualisations to maps that can be viewed in Web-based maps or in virtual globe programs, such as Google Earth. In order to facilitate the visualisation with KML we first generated a KML API based on the KML schema files with means of the Java Architecture for XML Binding (JAXB) [26]. Based on this API a class KMLWriter generates KML files visualising the results in form of 3D objects, which act as markers for those significant places that are relevant to a specific query in Google Earth. The KMLWriter is capable of three types of visualisation: SphereVisualisation, BarVisualisation, and CyclinderVisualisation. The SphereVisualisation takes a number of GPS coordinates, each with a corresponding integer value as parameter and generates a KML file that draws a semi-transparent sphere with a radius of the corresponding integer value for all coordinates. Based on the same input parameters, the BarVisualisation generates a KML file that draws one white bar at the position of the coordinates with the highest integer value, and grey bars for all other coordinates. The highest integer value represents 100 percent. The integer values of the grey bars are indicated by a dark filling with a height according to the percentage. Finally, the CyclinderVisualisation expects a number of GPS coordinates, but each with a corresponding array of 24 integer values as parameter. The highest integer value taken from all arrays represents 100 percent. This visualisation draws a cylinder at all coordinates, subdivided in 24 segments in the form of a pie chart. The 24 values from each array in percent define the colour of the segments ranging from light for 0 to dark for 100. These three visualisation types are used by the different queries. The output KML files are then visualised in Google Earth.

The SphereVisualisation is used to present the results of the InferenceEngineCountingPlaces and the InferenceEngineCountingTimes. The list of all significant places and their coordinates with either the overall number of stays or the overall time of stays per significant place is used as the input. The value of the overall number of stays or respectively the overall time of stays is used to define the radius of each sphere. BarVisualisation is used for the presentation of places a user is most probable at a given hour and the most probable follow up stays. The NextStateSensor passes the weekday and the hour chosen, from popup menus in the LocaRhythmsClientGUI to the InferenceEngineJAHMMLocation via XML-RPC in form of a SensorEvent. There a query to the relevant HMMs is prepared from the information on weekday and hour and the probabilities are inferred from the JAHMMLocationHMM. The results are weighted, by combining the probabilities of being at a specific place at a specific hour with the probabilities of being at a specific place at that specific day with different weights. Then, the weighted final result is sent back to the LocaRhythmsClient and used as parameters for the BarVisualisation. Finally, the CyclinderVisualisation is used for illustrating the stay probabilities for a specific time span for all relevant significant places. A weekday and a hour for the start and end is sent to the InferenceEngineJAHMMLocation, where again a query to the relevant HMMs is prepared that retrieves the weighted results for all inferred significant places between the specific hours for the specific days. The result, handed back in the form of SensorEvent, is then again used as the input parameters for the visualisation. For each query type and visualisation, the LocaRhythmsGUI finally displays the KML file in the Google Earth application.

IV. RELATED WORK

Begole et al. [4] analysed where users go online, and inferred rhythms. They only distinguish between three locations (office, home, lab) and they only visualised the collected data without probabilistic reasoning. Ashbrook and Starner [3] used a variant of the k–means clustering algorithm to identify places and a Markov model approach to predict the next possible place a person might visit. In contrast to PRIMILocaRhythms they do no support time prediction as only one global model exits for one user. Furthermore, the models are not updated in real–time. Eagle and Pentland [5] use cell ID and Bluetooth on mobile phones for positioning, and so are limited to a more coarse grained location precision outdoors, but can predict where a “low entropy” subject will be in the next hour based on the current context with an accuracy up to 90% with HMMs. This is not built in end-to-end architecture—the analysis of the data is only done on the collected data. Adams et al. [1] use a density-based clustering algorithm to infer ‘social spheres’ (i.e., significant places) from GPS traces and infer rhythms from this data. The data is not used for prediction but for a context-sensitive video and photo browser and blog. In comMotion Marmasse and Schmadt [15] identified buildings as a significant place for a user if the GPS signal was lost and after a time reappeared in a certain radius. They made tests with pattern recognition tools to predict the place a user is likely to go based on five different GPS routes. Liao et al. [14] use hierarchical Markov model to learn significant locations from GPS data logs but they omit to consider time of the day and the day of the week into their inference.

LocaRhythms integrates continuous capturing of data, real-time learning of new significant places with updated prediction models, and presenting information to users at anytime. Most of the related work has solutions for one or two of the above-mentioned challenges, but not a fullintegrated solution based on a modular design like LocaRhythms that works real-time.

V. CONCLUSIONS

We presented LocaRhythms, an approach for capturing GPS traces, extracting significant places and predicting and presenting probable stays in a real-time distributed application. Informal evaluations showed that prediction works well for users with constant rhythms (e.g., during a workday). For users with variations in their rhythms, a future version should reflect this uncertainty in the presentation. Furthermore, for real-world applications, we found that the battery consumption is a limiting factor for continuous capturing. Tests on the quality of prediction in dependency of a lower sampling rate of GPS coordinates will help us to find a better balance. We will focus on the detection of changes in long-term rhythms, as those changes are reflected to slowly. One of the challenges is to detect changes at different temporal granularity levels in order to model changes that only exist for a short period (e.g., holidays) adequately. Different spatial granularity levels can be explored to make better predictions on coarser grained movements (e.g., between cities). This should also help improving the functions for weighting between the predications of different models.

ACKNOWLEDGEMENTS

We thank the members of the Cooperative Media Lab; special thanks to Dennis Braunsdorf, Mike Drexel, Jan Hanak and Tobias Pohl. Thanks for the anonymous reviewers for comments on earlier versions of this paper.

REFERENCES

1. Adams, B., Phung, D. and Venkatesh, S. Sensing and Using Social Context. ACM Transactions on Multimedia Computing, Communications, and Applications 5, 2 (Nov. 2008). pp. 11:1-11:27.
2. Ashbrook, D. and Starner, T.E. Learning Significant Locations and Predicting User Movement with GPS. In Proceedings of the 6th IEEE International Symposium on Wearable Computers - ISWC 2002. IEEE Computer Society Press, 2002. pp. 101-108.
3. Ashbrook, D. and Starner, T.E. Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. Personal and Ubiquitous Computing 7, 5 (Oct. 2003). pp. 275-286.
4. Begole, J.B., Tang, J.C., Smith, R.B. and Yankelovich, N. Work Rhythms Analysing Visualisations of Awareness Histories of Distributed Groups. In Proceedings of the ACM 2002 Conference on Computer-Supported Cooperative Work - CSCW 2002. ACM, 2002. pp. 334-343.
5. Eagle, N. and Pentland, A.S. Reality Mining: Sensing Complex Social Systems. Personal and Ubiquitous Computing 10, 4 (May 2006). pp. 255-268.
6. Facebook. Facebook | Home. http://www.facebook.com, 2009. (Accessed 4/6/2009).
7. Francois, J.-M. Jahmm - An Implementation of HMM in Java. http://www.run.montefiore.ulg.ac.be/%7Efrancois/software/jahmm/, 2009. (Accessed 4/6/2009).
8. Google. Google Earth. http://earth.google.com/, 2009. (Accessed 4/6/2009).
9. Gross, T., Egla, T. and Marquardt, N. Sens-ation: A Service-Oriented Platform for Developing Sensor-Based Infrastructures. International Journal of Internet Protocol Technology (IJIPT) 1, 3 (2006). pp. 159-167.
10. Gross, T., Stary, C. and Totter, A. User-Centered Awareness in Computer-Supported Cooperative Work-Systems: Structured Embedding of Findings from Social Sciences. International Journal of Human-Computer Interaction 18, 3 (June 2005). pp. 323-360.
11. Jones, Q., Grandhi, S.A., Terveen, L. and Whittaker, S. People-To-People-to-Geographical-Places: The P3 Framework for Location-Based Community Systems. Computer Supported Cooperative Work: The Journal of Collaborative Computing 13, 3-4 (Aug. 2004). pp. 249-282.
12. Juang, B.H. and Rabiner, L.R. Hidden Markov Models for Speech Recognition. Technometrics 33, 3 (Aug. 1991). pp. 251-272.
13. Kang, J.H., Welbourne, W., Stewart, B. and Borriello, G. Extracting Places from Traces of Locations. ACM Sigmobile Mobile Computing and Communications Review 9, 3 (2005). pp. 58-68.
14. Liao, L., Fox, D. and Kautz, H. Learning and Inferring Transportation Routines. In Proceedings of the Nineteenth National Conference on Artificial Intelligence - AAAI 2004. AAAI Press, 2004. pp. 348-353.
15. Marmasse, N. and Schmandt, C. A User-Centered Location Model. Personal and Ubiquitous Computing 6, 5-6 (Dec. 2002). pp. 318-321.
16. Merriam-Webster, I. Merriam-Webster Online. http://www.m-w.com/, 2009. (Accessed 13/3/2009).
17. NIMA. Department of Defense - World Geodetic System 1984. National Imagery and Mapping Agency, ftp://164.214.2.65/pub/gig/tr8350.2/wgs84fin.pdf, 1984. (Accessed 29/6/2007).
18. NMEA. NMEA. National Maritime Electronics Association, http://www.nmea.org/content/nmea_standards/nmea_083_v_400.asp, 2008. (Accessed 4/6/2009).
19. OGC. KML | OGC. Open Geospatial Consortium Inc., http://www.opengeospatial.org/standards/kml/, 2008. (Accessed 4/6/2009).
20. Patterson, D.J., Etzioni, O. and Kautz, H. The Activity Compass. Presented at First International Workshop on Ubiquitous Computing for Cognitive Aids - Ubicog 2002. 2002.
21. Rabiner, L. and Juang, B. An Introduction to Hidden Markov Models. IEEE ASSP Magazine (Jan. 1986). pp. 4-16.
22. Sanchez, D., Tentori, M. and Favela, J. Activity Recognition for the Smart Hospital. IEEE Intelligent Systems 23, 2 (Mar.-Apr. 2008). pp. 50-57.
23. SiRF. Welcome to SiRF Technology. SiRF Technology Inc, http://www.sirf.com/products/gps_chip.html, 2009. (Accessed 4/6/2009).
24. Sun Microsystems Inc. JSR-000082 JavaTM APIs for Bluetooth - Final Release. http://jcp.org/aboutJava/communityprocess/final/jsr082/, 2004. (Accessed 4/6/2009).
25. Sun Microsystems Inc. JSR-000179 Location API for J2ME - Final Release. http://jcp.org/aboutJava/communityprocess/final/jsr179/, 2005. (Accessed 12/3/2009).
26. Sun Microsystems Inc. jaxb: JAXB Reference Implementation. https://jaxb.dev.java.net/, 2006. (Accessed 4/6/2009).
27. Sun Microsystems Inc. Developer Resources for Java Technology. http://java.sun.com/, 2009. (Accessed 4/6/2009).
28. Twitter. Twitter: What are you doing? http://twitter.com/, 2009. (Accessed 4/6/2009).
29. Westeyn, T., Brashear, H., Atrash, A. and Starner, T. Georgia Tech Gesture Toolkit: Supporting Experiments in Gesture Recognition. In Proceedings of the 5th International Conference on Multimodal Interfaces - ICMI 03. ACM Press, 2003. pp. 85-92.
30. Winer, D. XML-RPC Specification. http://www.xmlrpc.com/spec, 1999. (Accessed 4/6/2009).
31. Witten, I.H. and Frank, E. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers, San Francisco, 2006.