Time Based Context Cluster Analysis for Automatic Blog Generation

Luca Costabello *
Politecnico di Torino
Corso Duca degli Abruzzi, 24
Turin, Italy
luca.costabello@studenti.polito.it

Laurent Walter Goix
Telecom Italia Lab
Via Reiss Romoli 274
Turin, Italy
laurentwalter.goix@telecomitalia.it

ABSTRACT

This paper describes the algorithms developed to identify the actions of a community of users. Actions are detected through the processing of raw context data acquired from the users' smartphones or PDAs by our context awareness platform. Data mining techniques, and in particular cluster analysis, allow us to discover high level information, like the actions performed by the subjects during each day. This leads to the detection of a community of users, linked to each other by the places they have visited and by whom they have met.

Natural text description, enriched with Microformats semantic markup, is created for each user's action. Pictures and movies acquired during the day are added to the text, that is automatically published as a blog post.

Clustering algorithms explanation is provided, and the main issues of mining context data are addressed. A brief description of the blog generation is presented, too. Moreover, the perception that the users have of their own daily activities is analyzed, based on our experimental results.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering; H.3.4 [**Systems and Software**]: Distributed systems; H.3.5 [**Online Information Services**]: Data sharing,Web-based services

General Terms

Design, Algorithms, Experimentation

Keywords

Context awareness, cluster analysis, semantic web, blog

1. INTRODUCTION

The widespread popularity of personal weblogs is one of the key aspects of the contemporary web. A vast number of blogs includes personal *context* information, such as where the user has been or what actions he performed during the day. This data is typically enriched with pictures or videos acquired with smartphones or PDAs. Our automatic blog generator carries out the task of creating and publishing users' daily journals. Detailed personal blog posts describing each day are therefore created without the need for user intervention. The application allows to detect new communities according to the users' location patterns. Personal daily actions can be therefore compared to what other people did. (e.g. a commuter can find out that other users travel between home and work on the same road).

First of all, an overview of our system architecture is presented in Section 2.

In Section 3 we describe how we exploit each user's context information. This data is provided to our application by the Context Awareness Platform, the software layer that retrieves, organizes and stores the information from the users' devices.

Mining raw data leads to the discovery and the creation of a community of blogs, whose users are linked to each other by the context in which they have been. Section 4 describes the main issues of detecting users' actions.

Section 5 describes the offline processing of context information using datamining techniques. Our clustering algorithms are therefore explained.

The clustering algorithms detect what *actions* the subjects did during the day. In Section 6 these situations are converted into natural text to create a community of user-generated blogs. Each blog describes what the person did during the days, and it is generated automatically.

Section 7 presents the results of our trials and a few considerations about the users' feedbacks.

2. SYSTEM ARCHITECTURE

Blog post generation includes several steps, each of them performed by a specific module, as seen in Figure 1.

User's daily data must be fetched from the Context Awareness Platform. Retrieved information is processed by clustering algorithms. This step detects the actions performed by the user during the day (where he/she has been, how long, with whom, etc...). Algorithms can detect user movements, too.

At this point, blog posts need to be created. Clusters must be presented to the users in an appropriate way. We chose therefore to convert them into natural text, using a text generator. Movies and pictures acquired by the user inside the clusters are added, too. Multimedia content is retrieved via RSS from a CMS that stores user generated images and videos. Blog posts are eventually published on

*The work presented in this paper was done while the author was an intern at Telecom Italia Lab.

WWW2008, April 21–25, 2008, Beijing, China.
.

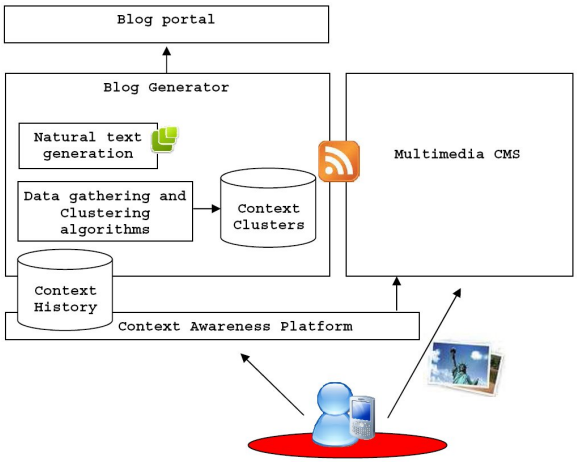

Figure 1: System architecture

the web, using a third party blog publishing system.

3. GATHERING CONTEXT DATA

The term *Context* typically identifies location, nearby people (or nearby objects) and, more in general, the surrounding environment [4]. Our work focuses mainly on user location. We work with many location technologies, such as Bluetooth and GPS, but most of our position data is cell-based. We concentrate on cell-based location, since the adoption of GPS devices is still not so widespread.

Other work has been performed to detect the instant situation of a user [3], but the approach is not suitable for the analysis of a whole day.

Detecting the situations of mobile users is possible only after the context data gathering step. Raw data needs to be processed and typically comes from heterogeneous sources, such as on-device sensors (e.g. Bluetooth). Operator-owned information needs, on the other hand, to be fetched from the network (i.e. GSM/UMTS cell IDs are retrieved in a timely fashion to determine user locations). Our Context Aware (CA) Platform processes data from the client application running on user devices and adds network related information [8]. Temporal dimension is considered, as it allows the creation of a so called *context history*. Starting from the users' raw context data logs, it is possible to perform datamining techniques in order to retrieve high level information.

3.1 Harvesting contextualized data

Getting each user's context data is the first step needed to generate a blog post. Data fetched from the *context history* contains raw information from heterogeneous sources, and needs to be processed by ad hoc CA Platform providers first [8]. Each record includes a timestamp, the GSM cell ID associated with the device, the user defined labels assigned to personal locations (*Virtual Places*) and a list of nearby CA Platform registered users. The platform provides both user names and relationships, since each nearby user is associated to his device's Bluetooth address.

3.2 Context data main issues

Each device performs updates to the context aware platform with variable frequency. This occurs when context changes, i.e. when the cell phone connects to a different base station to transmit the new GSM/UMTS cell ID. Clients refresh their context periodically too, even if no change has occurred. Refresh frequency is an user-defined parameter and it typically lasts a few minutes [8]. Discriminating static situations from dynamic ones is much easier with short update frequencies. Moreover, users may turn their devices off (or shutdown the clients) whenever they want. Clustering algorithms must deal with all these issues.

Cell based data follows different patterns according to *where* the user is located. Base station concentration is higher inside urban areas, where cells have a few hundred meter span, providing a more accurate user location. On the other hand, inside a dense populated area, base stations overlap and clients may experience frequent cell handovers, even if the user is not on the move. Detecting a static situation is, in this case, a more complex task. On the other hand, detecting fast movement is easier (e.g. user driving). A slow moving user (e.g. a passer by) generates patterns that can be confused with a static situation affected by many cell switches.

Rural areas are typically covered by a small number of base stations, spanning a few Kilometers. Location is therefore less fine-grained. In these circumstances, static actions are easier to detect, since there are fewer overlapping cells. On the other hand, intra-cell movements cannot be detected, since no cell handover is performed.

4. IDENTIFYING USERS ACTIONS

The goal of our software is the creation of a web community where users share their context data through the publication of their daily situations enriched with geo-referenced pictures and videos. The blog generator's clustering algorithms are able to detect and extract what kind of *actions* the users have done during the day, trying to carry out the task in the most accurate way (Figure 2). By doing so, a community of interconnected blogs is automatically created: users can browse their daily posts and find out what their friends did or what pictures they took. Clustering process focuses on *location* and *time* dimensions: chronological order of events is therefore respected.

The context blog generator must detect when the users are

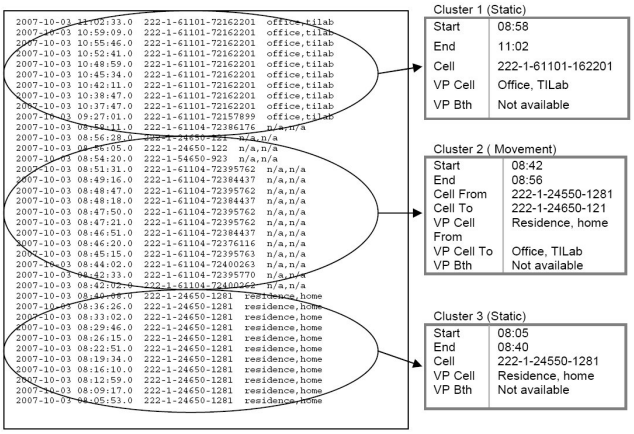

Figure 2: Context data clustering process. Two static situations and one movement cluster have been detected.

moving or not. It's therefore necessary to define two classes of clusters, the *static* clusters and the *dynamic* ones. *Static* clusters own timestamp information of the beginning and of the end of the cluster, the GSM-UMTS cell ID chosen as center, and the Virtual Place labels, (if any) assigned to it or to a fixed Bluetooth beacon. Besides timestamps information, a *dynamic* cluster includes the cell IDs and the Virtual Place labels (if present) of both starting and arrival locations. A Bluetooth Virtual Place, if present, identifies the carrier (e.g. car, bus, train). Both cluster classes include a list of other nearby users.

5. CLUSTERING ALGORITHMS

Clustering process works on location data. Both GSM-UMTS cell ID and Virtual Place labels are strings. Distance-based clustering algorithms (e.g. K-Means, [9]) are therefore not suitable for the task, since it is not possible to define the euclidean distance for cell IDs or user-defined labels. Furthermore, working with existing algorithms that focus specifically on categorical attribute (e.g. ROCK [5]) would be incomplete, because our solution has to deal with time, to respect the chronological order of the events.

This led us to design ad hoc algorithms to solve the problem. We describe here two different approaches to the same issues, Compare&Merge and Multi Level Sliding Window. They differ in terms of outcome and are to be used in different situations, as seen in Section 5.3.

5.1 Compare&Merge algorithm

The algorithm evaluates records in inverted chronological order (from the most recent to the oldest). An average day is typically made of hundreds of context records. It is possible to group data according to GSM-UMTS cell IDs or to Virtual Place labels, if set by the users. User defined labels allow the algorithm not to deal with cell ID issues (Section 3.2). Data clustering procedure works as follows:

1. *Context History preliminary scan.* Temporally near records with the same locations are merged (Figure 3). This step detects non-ambiguous static situations, time spans during which the user has performed several context updates with the same location (i.e. same cell IDs or same Bluetooth/cell ID labels). Our trials showed that these are the most common situations, therefore they must be detected properly.

   At the end of the scan we get a list divided into *long temporary clusters*, each of them clearly marking a static situation, and *short temporary clusters*. The latter can be part of a movement situation, but the user could have been affected by spurious cell handovers instead (Section 3.2). This ambiguity is solved in step 2.

```
foreach ContextEntry ce{
  if (ce.location == clusterTemp.location)
    clusterTemp.addContextEntry(ce);
  else{
    tempList.add(clusterTemp );
    clusterTemp = new Cluster(ce);
  }
}
return tempList;
```

Figure 3: Context History preliminary scan

2. *Temporary clusters merge* (Figure 4). This step may create movement clusters.

   The cluster list obtained at step 1 is processed. Particular attention is given to short temporary clusters: these are merged in a wider cluster matching the original chronological order. Before being added to the final list, this newly created cluster is analyzed. In case it includes too many different locations (a proper threshold must be set), it is labeled as a movement cluster, otherwise it is a static one. In this case, the most frequent location in the cluster is set as the cluster center.

```
foreach Cluster currCluster{
  // if cluster is too short
  if (currCluster.duration < durationThreshold){
    tempCluster.extend(currCluster);
  }
  // if cluster is long enough
  else{
    // if there are enough different locations
    if (tempCluster.locationsCount > locationsThreshold)
      tempCluster.moving = true;
    finalList.add(tempCluster);
    finalList.add(currCluster);
  }
}
return finalList;
```

Figure 4: Temporary clusters merge

Figure 4 shows two important parameters:

`durationThreshold` is the minimum duration required for a cluster to be added to the final list. The threshold filters too short actions, that are merged in a wider cluster instead. This parameter controls clustering detail level.

`locationsThreshold` is the maximum number of different locations that a static cluster can include. Beyond this threshold the cluster is labeled as a movement situation. `locationsThreshold` can be influenced by the base station density (Section 3.2) and by the users' perception, as seen in Section 7 (Detecting user movements).

The algorithm loses its effectiveness when extensively used in non-labeled areas affected by frequent cell handovers, as seen in Section 3.2. In this case, Compare&Merge might erroneously label a cluster as dynamic if the latter owns more than `locationsThreshold` different cells. To overcome this problem, a brand new algorithm has been developed (Section 5.2).

5.2 Multi Level Sliding Window algorithm

Compare and Merge is challenged by long-lasting, non-labeled context data sequences affected by frequent cell handovers. These patterns might be confused with movement actions, even if the user is not moving. The problem can be lessened by marking the switching cells with the same user-defined label, but such workaround cannot be used in all cases (obviously, users cannot label every visited place).

MultiLevel Sliding Window algorithm solves the issue, without changing input data format. Clustering is performed in a hierarchical way, starting from user labels based on Bluetooth or on cell ID. If no labeling information is found, the algorithm will work on GSM-UMTS cell IDs. Context data is processed in chronological order, to comply with the actions' chronology.

A sliding window is used to process each user's context history (Figure 5).

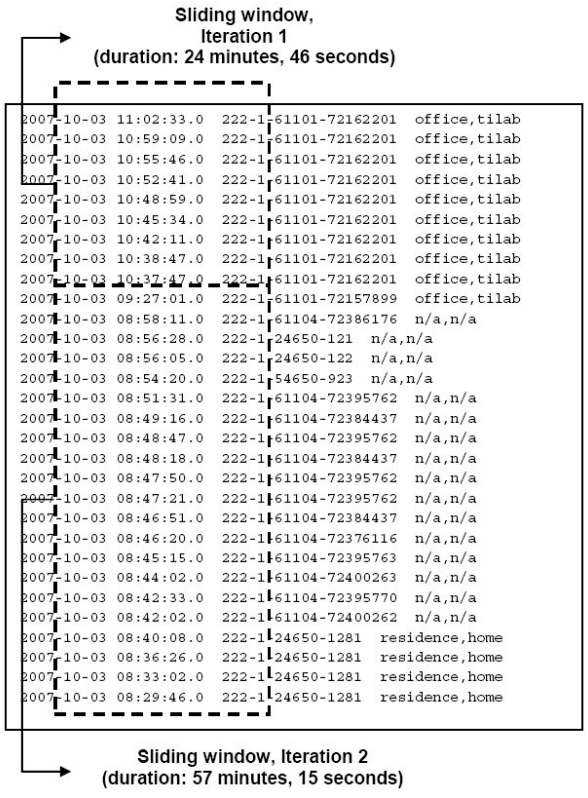

Figure 5: Sliding window example

The window has a fixed *maximum* length, expressed in minutes by a specific parameter. in Figure 5 the window can be at most 60 minutes long. The first context record has been generated at 11:02:33. Our window cannot therefore include records before 10:02:33. Sample data lacks records from 10:37:47 to 09:27:01 (the user might have turned the device off). The last record to be included inside the window is therefore the one uploaded at 10:37:47, since the algorithm does not infer user position when no context update is available. Therefore, the first iteration, produces a 24 minute long window. The second iteration causes the window to slide: the first valid record is at 09:27:01 and, as explained above, the last is the one updated at 08:29:46.

The algorithm works therefore in a time related fashion, without being aware of how many records are included inside each iteration's window. Context data records occur with non deterministic frequency, as explained in Section 3.2: tuning the window length on record number would have been incorrect. Nevertheless, the window's duration does depend on user-defined context data update policies: the window must include enough records in order to obtain better results. If context updates occur with low frequency, the window's duration needs to be extended.

Data can be merged in compliance with the location hierarchy (this is where the *MultiLevel* adjective comes from). Highest priority is given to Bluetooth Virtual Places, since they typically identify small areas (e.g. rooms, offices). If none of these is set, the algorithm works at cell ID Virtual Place level. If no user label is set, the procedure works at cell ID level. However, the number of nested levels and the related data bindings can be edited, in order to use the algorithm in different scenarios.

Figure 6 shows the algorithm core method. Having created the desired window, the included context data records are processed with the following functions:

- `detectLevel()`: sets the current working level for each window. In order to do so, window context records need to be analyzed: if a certain number of Bluetooth Virtual Places is available, the algorithm will work at this level, otherwise the lower levels are exploited in the same way. The lowest level, (GSM cell IDs) is always available. The algorithm goes down a level according to an empirically chosen threshold.

- `detectMovement()`: detects a dynamic situation in the window's records, working at cell ID level only. Indoor motion, identified by Bluetooth ID changes, is not important to blogging purposes. Movement is detected as follows: each cell ID is given a counter. If no cell ID recurs for more than 33% of total records there is a dynamic situation (the threshold has been tuned with empirical tests, see Section 7).

- `detectCentre`: the method detects the most frequent position among the window's records. The function works at the previously selected level and can return a Virtual Place or a cell ID. At first, 70% of the records inside the window had to own the same position, but the percentage was excessively high due to cell handovers issues (Section 3.2). Empirical tests showed that one third (33%) of records at the same location level is enough to select the window's center.

After each iteration, the window's records are merged into the temporary cluster only if the window's center equals the temporary cluster's one. Before expanding the temporary cluster, a further test is performed: if too much time has elapsed since the window beginning, the merging is forbidden (Figure 6, `offlineTimeSpan`). It is perfectly sensible

```
// create initial window
List windowElements = getWindowElements(0);
// iterate over context records
while (windowElements.size() > 0){
  level = detectLevel();
  isMoving = detectMovement();
  centre = detectCentre(level);
  if ( compareCentres(centre,cluTemp,level) &&
       getGapTimeSpan(cluTemp,windowElements)< offlineTimeSpan){
    // extend current temporary cluster
    cluTemp = extendCluTemp(windowElements);
  }else{
    // save temporary cluster
    clusterList.add(cluTemp);
    // create new temporary cluster
    cluTemp = convertToCluster(windowElements);
  }
  // window sliding
  windowElements = getWindowElements(windowElements.size());
}
return clusterList;
```

Figure 6: MultiLevel Sliding Window, core method

that a long period without receiving context updates leads to the creation of a new cluster. Inferring the user location for such a long time would be wrong, as shown previously in this Section. Tests showed that a reasonable value for this time gap is one hour.

5.3 Algorithms comparison

Compare & Merge's most important features are briefly listed below:

- high clustering precision in case of extended user locations labeling. Actions are correctly detected. Temporal uncertainty added by the procedure is irrelevant.
- user movement detection, i.e. dynamic clusters.
- Short lasting actions can be easily discarded via the `durationThreshold` parameter.

MultiLevel Sliding Window's main features are listed below:

- The algorithm solves cell handovers issues, even with a high number of overlapping cells, thanks to the sliding window. It is therefore suitable for poorly labeled areas.
- Movement detection.
- Flexible number of location levels.
- Precision is inferior to the one guaranteed by Compare & Merge: large windows lead to higher uncertainty.
- Brief actions can be ignored, but this is achieved by tuning the window dimension. The task is therefore less immediate, since there is not an explicit minimum duration threshold.

Table 1 summarizes the clustering algorithms' main features. Execution time is not a crucial issue, since daily context data is made of a few hundreds records only for each user.

6. AUTOMATIC BLOG GENERATION

Clustered data has to be converted into natural text, in order to create the user blog post (Figure 7).

First of all, standard sentence structures have to be defined (e.g. sentences for both static and dynamic situations). Suitable verbs must be chosen, according to the presence of movement or to what type of Virtual Place is set (e.g. *to work* if the user has been in his office, *to shop* for a mall or another shopping area, *to move* for dynamic clusters, etc) (Figure 7). Text detail can be customized too (e.g. nearby users can be showed or not, action durations can be hidden, etc...). Text is built using a third-party natural text realiser, SimpleNLG [12], which eases the sentence construction (e.g. it handles trivial punctuation, paragraphs division).

Many sentence blocks such as civil addresses and nearby people are enriched by the use of Microformats semantic markup techniques [1]. Microformats make mashing up easy: with a parser-equipped browser it is possible to view users' locations in maps or to add nearby buddies as contacts inside email clients. Microformats let us create structured blog posts: civil addresses and Virtual Places are embedded inside `adr` Microformats. A combination of `hcard` and `XFN` is used to include other nearby users. The former is needed to include all the additional information of the person retrieved from the user profile hosted on the Context Awareness Platform (e.g. company name, telephone number, email address, url...). `XFN` is used to introduce relationships, instead. Nearby buddies can be related to the blog owner in different ways, and this information is included in each post. It is therefore possible to navigate through users blogs, since each `XFN` Microformat links to its user's personal web log. Embedded Microformats own additional information both about location and nearby buddies: this data is hidden using an ad hoc CSS, in order not to spoil the structure of sentences (Figure 7).

Blog posts are enriched with contextualized multimedia content. Pictures (and movies) taken during the day are embedded inside text and are assigned to the context in which they have been acquired. Multimedia files are retrieved via RSS feeds from our multimedia CMS.

Blog Posts are eventually published on the web, using a third-party blog publishing platform. Figure 7 shows an example of published blog post.

The blog generator is implemented as a standalone application. Context Aware Platform interoperability is achieved through RESTful web services. The program runs in batch mode, and it is scheduled daily at 5:00 am. User days are assumed to start at the same time. At this time most users are still asleep and, on the other hand, night actions after midnight can be included in the previous day, even if they are part of a new one (many users still perceive them as part of the previous day).

7. RESULTS

Context data cluster analysis is connected to user habits. Our blog generator, using clustering techniques, aims at reconstructing what the subjects did during each day in the most accurate way. Accuracy means understanding the users' actions perception in order to deliver an output similar to their memories. The software has been tested on a group of ten users for over six months. The trial helped us tuning the following features:

Cluster analysis output. Figure 8 presents two different sample days, analyzed with the developed algorithms. The outcome has been compared to the user perception of his actions (each situation is labeled with a letter). Compare & Merge has been set to discard the events shorter than 30 minutes, while MLSW has been given a 30 minute

	Compare & Merge	Multilevel Sliding Window
Type	Categorical attributes with time handling	
Input data	GSM/UMTS cell ID, Bluetooth, user-defined labels	
Movement detection	yes	
Optimal usage	Good user labeling, GSM cells with low handovers issues	Days spent in non-labeled areas
Critical situations	Frequent cell handovers	none
Precision	High in optimal situations (<context update ≈ 2-5 minutes)	Variable. A 30 minute window leads to small errors (<30 minutes)
Brief situations filtering	User-defined	Sliding window length related.

Table 1: Clustering algorithms' main features

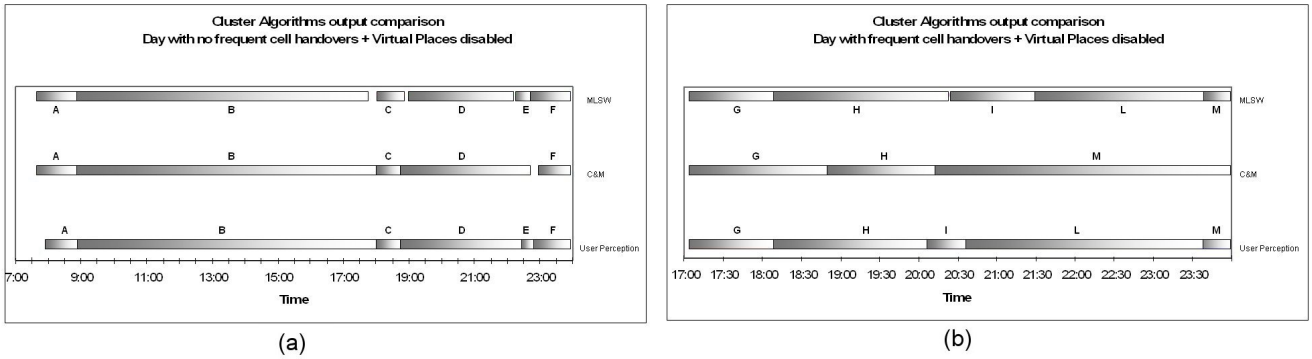

Figure 8: Cluster algorithms output comparison. Each situation is labeled with a different letter. Compare & Merge `durationThreshold` and MLSW's window are both set to 30 minutes. In (a) both algorithms detect the daily situations (Compare&Merge discards situation E, that is too brief). In (b) Compare&Merge does not follow correctly the day affected by frequent cell handovers, while MLSW does.

long window. Figure 8.a compares the outcome of Compare&Merge and MultiLevel Sliding Window in a day without frequent cell switches. Both algorithms detect the right clusters, and C&M has a higher precision than MLSW (Figure 9 shows the error analysis and a detailed description is provided later on in this section). Situation E has not been detected by C&M, since it is shorter than 30 minutes; MLSW does not discard situation E since the algorithm is less precise on filtering brief actions (Table 1). In Figure 8.b the algorithms have been tested on a day affected by frequent cell handovers (the user spent some time in downtown Milan). In this case, C&M's output is wrong, since situation L is not detected at all and it is confused with situation M. MLSW, on the other hand, detects all the situations.

Error analysis. Final error is built of two distinct components. Technologies used by the Context Aware Platform to locate people are affected by intrinsic error (e.g. movements inside a GSM-UMTS cell cannot be detected by our software). User-defined labels (Virtual Places) can act as another error source (e.g. leaving the Virtual Place *Office* typically means going out of the associated GSM cell, but the latter may span a considerable wider area).

Another error component comes from the clustering algorithms. Compare&Merge guarantees high precision in presence of user-defined labels, while GSM cell ID location causes a few minute error (in presence of frequent cell handovers there could be problems, as seen in Section 5.1). MultiLevel Sliding Window error is strongly connected to the window's duration. Empirical tests led to choose a 30 minute long window, in order to reduce the uncertainty to a few minutes. Longer windows tend to generate higher errors.

It is important to underline that users have quite a rough memory of their daily actions. Generally speaking, an average error of 10-15 minutes for each cluster allows to fulfill the tracking of user activities.

Figure 9 shows the clustering algorithms error analysis for the context data collected for a certain user on a day not affected by frequent cell handovers (Figure 8.a shows the related algorithms' output). A transition is the ending point of a cluster and the beginning of a new one. During the day the user noted when he started or finished a situation: these timestamps are compared with the clustering algorithms outputs. A *routine day* includes locations that the user has usually labeled as Virtual Places (e.g. home, office, shopping areas). It is therefore possible to evaluate the effect that user-defined labels have on the clustering procedure: the same data has been then processed with and without the Virtual Place labels. In the example, Compare & Merge has been set to discard situations shorter than 30 minutes. If a cluster owns 5 or more different cell IDs, C&M will mark it as dynamic. On the other hand, MLSW has been run with a 30 minute long sliding window. In the example, Compare & Merge (C&M) is more precise than MultiLevel Sliding Window and (with user labeling) leads to an average error of 6 minutes, while MLSW has an average error of 8 minutes. If we exclude user-defined labels, the clustering process is typically more difficult, and this is due to a higher number of GSM cell switches. In this case, MLSW is affected by an average error of 12 minutes, while Compare & Merge owns a 5 minute average error. In fig-

Figure 7: Sample blog post

ure 9, even without user labeling, Compare & Merge works well and guarantees small errors. This is because the user has not been in areas affected by many cell switches. In the latter case, the algorithm would not detect the right situations (Figure 8.b). On the other hand, despite being affected by higher errors, Multilevel Sliding Window detects user actions even with these problems.

Cluster minimum length. Algorithms have been tuned to discard short lasting situations. This occurs particularly when the user stays for a short time in rooms equipped with Bluetooth beacons. People tend to forget these repeated short events. The vast majority of tested users did not pay attention to situation shorter than 30 minutes. On the other hand, raising this threshold would lead to a significant detail loss. In Figure 9, transition 8 and 9 are only detected by MLSW, since the situation between them is too short and it is therefore discarded by Compare & Merge (Section 5.1).

User labeling role. Both clustering algorithms can work without user-defined labels (even if Compare&Merge might have problems with many cell switches). During the tests, it has been clear that the effectiveness of the merging procedures is higher when Virtual Places are set (an extensive use of customized labels reduces cell handovers problems, Section 3.2) (Figure 9).

Detecting user movements. A cluster is labeled as dynamic only if it includes a number of different locations greater than a threshold. To tune this parameter it is important to know how users perceive their movements. For example, many people do not consider strolling a real movement situation, but cluster analysis may, in this case, label the action as a dynamic situation. Moreover, data patterns vary according to movement speed: fast moving situations are identified by a clear sequence of different cell IDs, while slow ones (e.g. going out for a walk) may be ambiguous (Section 3.2).

According to our tests, in order to detect a dynamic situation, 5 different cell IDs are needed for Compare&Merge algorithm, while for MultiLevel Sliding Window, no cell ID must excess 33% of each window's records.

Inter-cluster gap. The CA platform may not receive context updates for a relevant time span. Devices might be turned off, or clients may not be running. Clustering algorithms allow a maximum time gap of 60 minutes. Beyond this threshold, a new cluster is created, even if the location has not changed.

8. RELATED WORK

Other works focused on automatic generation of user action logs have in part inspired our work.

Pepys system [10] is aimed at working as a human memory backup system, through the creation of a personal log. User location is collected from wireless badges. Situation discovery is achieved by observing user gatherings, while movement is detected by a dedicated algorithm. After a further generalization, results are converted into schematic text.

Mobilife Life Blogging [6] fetches context data from user devices (PDAs, smartphones), from which a client gathers raw information. Blog posts include context data and merging algorithms are used, but the succession of events is not considered.

ActionLog is an automatic log generator system, aimed at sharing information between users [11]. Location data is exploited to describe user actions. Users can edit generated text before publish it on their blogs, but this step is done as long as the action is performed by the user, so there is no data clustering.

Other works can be cited, even if not directly aimed at user log generation.

The MyCampus Experience [13] is a framework focused on providing context-aware services to a community of college students. One of these services allows the users to post and retrieve posters based on their interests and locations. The application detects the users' typical routes inside the campus, using wireless LAN location.

SPECTER [7] is a context aware personal assistant software that exploits action logs (Personal Journal) to aid the user in everyday life.

The Personal Journal is made of auto-generated entries, but users are involved in the process too. Personal Digital Secretary [2] works with both context and user profile data. In this case, context aware techniques are therefore supported by more traditional data retrieval. Data is merged by a proper module, in order to get high level information.

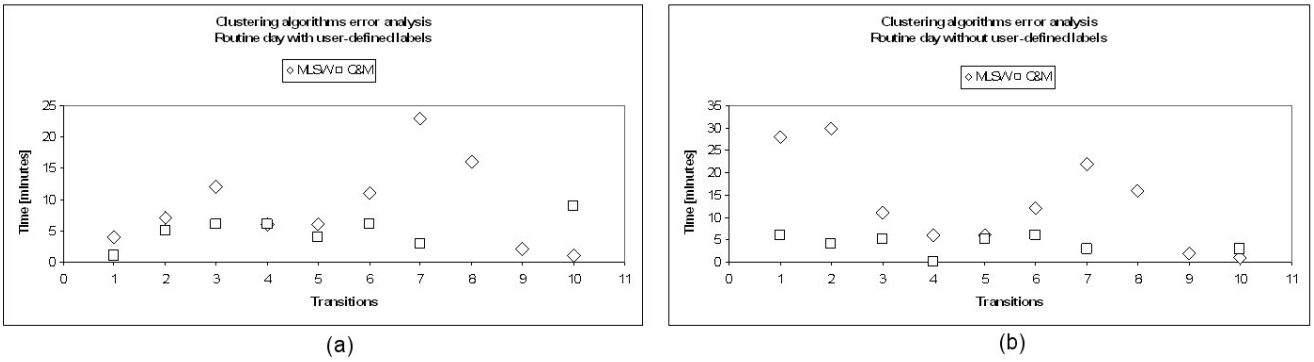

Figure 9: Virtual Places vs non labeled areas. The figure shows the error analysis for the same routine day. No frequent cell handovers are present. Compare & Merge `durationThreshold` and MLSW's window are both set to 30 minutes. Clusters analyzed in (b) are shown in Figure 8.a. Transitions 8 and 9 are not present in C&M outcome, since cluster E has been discarded (Figure 8.a)

9. CONCLUSIONS AND FUTURE WORK

This paper describes the most important features of our automatic blog generator. Our clustering algorithms let us group and merge user context data, with special concern to cell ID location information. To achieve this goal, context data issues have to be faced (Section 3.2). Two different algorithms have been designed, in order to fit different context data patterns (Section 5.1 and Section 5.2). Our trials with real life context data helped tuning algorithms parameters in order to minimize errors. Clustering process and natural text generation eased the analysis of how many daily activities' details the users remember. These facts have been exploited to design a blog post text structure and to choose the best detail level (Section 6).

Privacy is a crucial issue to deal with. A few users considered the software too intrusive, since many did not want their actions to be tracked and shared with others. Our prototype still does not include a privacy level selector for each user's blog posts. The feature will allow users to choose who can read their daily logs (e.g. private, friends only, public), and which kind of information display.

Daily batch schedule is still temporary: users will be allowed to select the starting time of their days. The same algorithms can be adapted to work with other location technologies, such as Wireless LAN. GPS location data could be used, too (in this case clustering process will be therefore distance based, so different algorithms have to be used). Blog posts are now isolated. Future developments will allow new posts to be aware of previous days history, in order to deliver a better user experience.

10. REFERENCES

[1] J. Allsopp. *Microformats: empowering your markup fow Web 2.0*. Friends of ED, 2007.

[2] H. E. Byun and K. Cheverst. Exploiting user models and context-awareness to support personal daily activities. *Workshop in UM2001 on User Modelling for Context-Aware Applications*, 2001.

[3] L. Cerami, L.W. Goix, M. Valla, and P. Falcarin. Situation inference for mobile users: a rule based approach. In *Proceedings of MCISME*, 2007.

[4] A.K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 2001.

[5] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. *In 15th International Conference on Data Engineering*, 1999.

[6] J. Koolwaaij, A. Tarlano, M. Luther, P. Nurmi, B. Mrohs, A. Battestini, and R.Vaidya. Context watcher - sharing context information in everyday life. In *Proceedings of WTAS*, 2006.

[7] A. Kröner, S. Baldes, A Jameson, and M. Bauer. Using an extended episodic memory within a mobile companion. In *Pervasive WS on Memory and Sharing of Experience*, 2004.

[8] L. Lamorte, C.A Licciardi, M. Marengo, A. Salmeri, P. Mohr, L. Roffia G. Raffa, M. Pettinari, and T. Salmon Cinotti. A platform for enabling context aware telecommunication services. *Third Workshop on Context Awareness for Proactive Systems*, 2007.

[9] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[10] W. M. Newman, M. A. Eldridge, and M. G. Lamming. Pepys: generating autobiographies by automatic tracking. In *Second conference on European Conference on computer-supported cooperative work*, 1991.

[11] K. Numaa, H. Takedab, H. Uematsuc, T. Nishimurad, Y. Matsuod, M. Hamasakid, N. Fujimurad, K. Ishidad, T. Hoped, Y. Nakamurad, S. Fujiyoshif, K. Sakamotof, H. Nagatag, O. Nakagawag, and E. Shinborig. A weblog grounded to the real world. In *AAAI-2006 Spring Symposium on Computational Approaches to Analyzing Weblogs*, 2006.

[12] Ehud Reiter. Simplenlg. http://www.csd.abdn.ac.uk/∼ereiter/simplenlg/.

[13] Norman Sadeh, Fabien Gandon, and Oh Buyng Kwon. *Ambient Intelligence: The MyCampus Experience*, chapter in Ambient Intelligence and Pervasive Computing. Eds. T. Vasilakos and W. Pedrycz, ArTech House, 2006.