High-level Automatic Event Detection and User Classification in a Social Network Context*

Fabio Persia[1] and Sven Helmer[2]

[1] Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
fabio.persia@unibz.it
[2] University of Zurich, Binzmühlestrasse 14, 8050 Zurich, Switzerland
helmer@ifi.uzh.ch

Abstract. We present a framework for high-level automatic event detection and user classification in a social network context based on a novel temporal extension of relational algebra, which improves and extends our earlier work in the video surveillance context. By means of intuitive and interactive graphical user interfaces, a user is able to gain insights into the inner workings of the system as well as create new event models and user categories on the fly and track their processing through the system in both offline and online modes. Compared to an earlier version, we extended our relational algebra framework with operators suited for processing data from a social network context. As a proof-of-concept we have predefined events and user categories, such as spamming and fake users, on both a synthetic and a real data set containing data related to the interactions of users with Facebook over a 2-year period.

Keywords: Event Query Languages · High-Level Event Detection · Intervals · Social Network Analysis · Behavior Identification in OSNs.

1 Introduction

In the era of big data we have to cope with continuously increasing data collections and data streams originating from various sources. Here, we look at logs containing user interactions with a social network. Usually, the persons (or systems) evaluating the data are not interested in looking at the enormous amount of raw events, but want to be informed about events on a more abstract level. For instance, a log of user activities may pick up every single click of a user, but when investigating the data we may be much more interested in detecting potentially harmful activities and user categories, such as *spamming* or *fake users*.

In fact, Benevenuto et al. [3] analyze user activities through *clickstream data*, initially focusing on statistical properties related to traffic and session workloads (e.g. access frequency, session duration, etc.). In a second phase, they employ a first-order Markov Chain to define a model of behavior describing dominant

* This work was supported by an internal grant from the Free University of Bozen-Bolzano under IN2078 (HAMSIK - High-level AutoMatic event detection in a SocIal networK context).

activities and transition rates between them. Schneider et al. [12] perform an analysis of clickstream data to identify typical user navigation strategies. They reconstruct clickstreams from anonymous HTTP header traces obtained from passively monitored network traffic with thousands of users from different Internet Service Providers and then apply a flexible methodology for identifying user sessions within the OSN. In addition, Amato et al. present a two-stage method for anomaly detection in the behavior of persons while using a social network [1, 2]. In a first step, a Markov chain model is used to automatically learn typically normal behavior of users. In a second step, an activity detection framework based on a possible worlds model is applied to detect unexplained activities deviating from the normal behavior.

Our approach models user behavior using complex events. Complex events are usually described in terms of individual simple events standing in a certain temporal relationship with each other. At the core of our system is a temporal relational algebra used to process high-level (and also medium-level) events. This allows us to tap into new efficient methods developed for processing data in temporal database systems [4, 11]. We deal with the complexity of high-level events by dividing our system into three layers. The lowest layer generates raw events, in our case related to individual time-stamped observation data depicting users' interactions with a social network (e.g., Facebook or Twitter). This layer is highly dependent on the application domain and has to be adapted if we want to move to another domain (we started from a video surveillance context [8], but we are also planning to apply our framework to data from other heterogeneous data sets, such as *Wikipedia*, *Yago*[3], or *GovTrack*[4]). The middle layer takes raw events and creates simple events whose format is largely independent of the application domain, thus separating the high-level event detection from technical details of the raw events. Additionally, events generated by the middle layer already contain some aggregated data, simplifying the high-level detection. Finally, on the highest layer a user can construct the complex events that they are really interested in, using medium-level events as building blocks. For ease of use, we also provide a graphical user interface (GUI) for formulating high-level events. The motivation of this paper comes from the fact that, to the best of our knowledge, there are no other interactive frameworks in the social network analysis context able to carry out the overall monitoring process from the lowest up to the highest layers, as well as to improve the support users receive in defining the event models they want to look for by means of smart graphical user interfaces.

As a result, a user can employ our system in a highly interactive way. In our demo, all the different parts of the event detection process on all three layers of the system can be observed in action and also be modified. Event detection can be run in two different modes. In the *offline* mode, a historical data set is analyzed after all the raw events have been generated and are stored, for example

[3] http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

[4] https://www.govtrack.us/

in a database. This is generally used for forensic purposes. In the *online* mode, the time-stamped low-level data is immediately processed by the system as it is generated. For our demo, we plan to use two data sets - a synthetic and a real one - that can be analyzed in the offline and online mode.

In summary, we make the following contributions:

- We present all three layers of a highly interactive event detection system, ranging from the generation of raw events to the formulation of complex high-level events.
- The system is based on an extension of relational algebra, ISEQL (Interval-based Surveillance Event Query Language) [6], enriched with powerful temporal operators.
- With respect to its previous version [6], we further enhance the expressivity of ISEQL, by introducing two new operators - *cardinality* and *overlap percentage* - and implementing them in the form of *PostgreSQL* stored procedures; such operators are particularly useful for defining high-level event models and user categories in a social network context.
- In the demo, we show the user interfaces of the system and also provide insights into the inner workings by allowing users to run event detection in an offline as well as an online mode.

2 System Architecture

The overall architecture of our proposed system is shown in Fig. 1. It consists of three layers: an *online social network (OSN) crawler*, an *interval action detector* and a *high-level event detector*. A similar architecture was the topic of earlier work [5, 6, 9, 8] in a video surveillance context. The output of the OSN crawler consists of a set Λ of collected data related to user sessions on a particular OSN. More specifically, examples of such interactions are a user "*FABIO*" who logged in at timestamp "2017-05-17 11:39:12", or a user "*SVEN*", who received a message at timestamp "2017-05-17 11:39:27" (Table 1). The interval action detector extracts medium-level events from Λ by labeling a sequence of OSN log entries with descriptors such as "Status&Friends" or "Shares". This layer produces as output a set M of medium-level annotations referring to intervals of entries within the log. Consequently, each user session at this semantic layer is modeled by means of a sequence of higher level intervals, rather than with a list of time-stamped low-level action symbols. Eventually, the high-level event detector takes a set E of event models and determines whether any of these events occur in M. Moreover, it also performs the user classification, thus assigning to each tracked user a category referring to a specific temporal interval. Thus, we assume the availability of a *log* describing a sequence of user interactions with an OSN. Our aim is to discover subsequences – in a log recording user activities – matching models of known user behavior and to perform an effective user classification (the formal definitions of *OSN Log*, *Interval Labeling*, *High-Level Events*, and *User Classifications* are given in [10]). More specifically, each of the

listed layers consists of three different sublayers (Fig. 2): a *graphical user interface (GUI)*, an *application core*, and a *database*. The *GUI*, implemented using *Java Swing APIs*, allows users to interact intuitively with the framework, guiding them along in a step-by-step manner, making our system usable for people with no prior knowledge in relational algebra. The *Application Core*, developed in Java, collects all the input coming from the *GUI*, checking it for correctness. On the other end, it stores *PL/pgSQL* versions of *medium-level* and *high-level* event models to make them persistent. It also invokes existing event models on a specified data set, detecting the specified events in the data set. *PostgreSQL 9.4* is the underlying *database* and every operator - including *cardinality* and *overlap percentage* - of both a *medium-level* and a *high-level* event model is implemented via stored procedures in a PostgreSQL DBMS. The individual operators can be assembled dynamically into different event models.

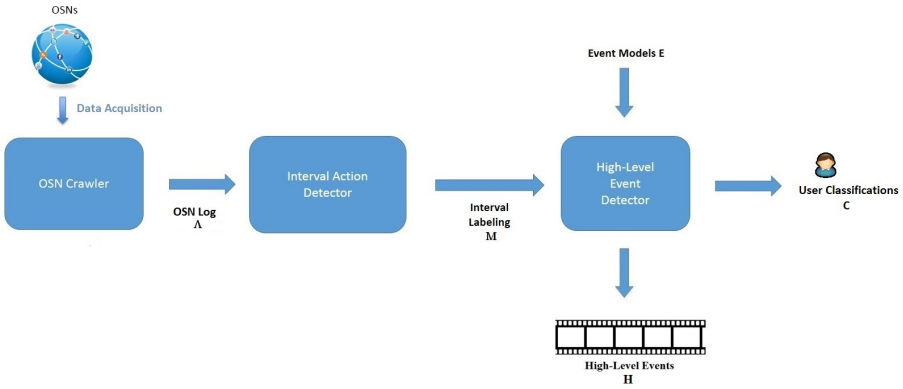

Fig. 1. Overall architecture

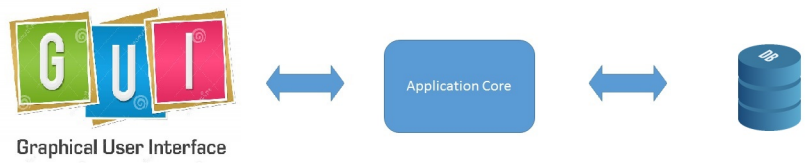

Fig. 2. Semantic sub-layers

2.1 OSN CRAWLER

In an earlier project, an OSN crawler to collect data from Facebook users was developed [1]. More specifically, this was done with PHP utilizing Facebook

APIs to capture data within secure sessions following the OAuth protocol. The application was then shared on Facebook and collected, after receiving authorization from users, data related to the interactions of about 1600 users over a two-year period from 2013 to 2015. The collected data was anonymized by assigning a random ID to each user and dropping personal details such as age and gender. Additionally, all participating users were informed that our work was purely research-related and that there were no commercial uses (falling under Facebook's privacy policy, which prohibits and punishes unlawful misuse). In the long run, the aim of the research is to protect users from fraudulent behavior. We are aware that technology like this can be misused, but this is true for security-related methods in general and needs to be discussed in a wider context [7].

Table 1 shows an example of a log obtained by the OSN crawler from Facebook (for the sake of readability we have replaced the random user IDs with our own names in this and the following tables). Table 2 gives an overview of the atomic user actions (along with their high-level categories) that we captured. Clearly, the OSN Crawler can also be used for capturing user interactions with other OSNs, such as *Twitter*; in that case, the content of the user tweets can also be collected and then stored.

Table 1. Example of OSN Log

Action Symbol	**User**	**Timestamp**	**IP**
login	FABIO	2017-05-17 11:39:12	192.168.1.88
likes a page	FABIO	2017-05-17 11:39:20	192.168.1.88
login	SVEN	2017-05-17 11:39:24	
message sent	FABIO	2017-05-17 11:39:27	192.168.1.88
message received	SVEN	2017-05-17 11:39:27	
status wall post	SVEN	2017-05-17 11:39:30	
message sent	SVEN	2017-05-17 11:39:40	
message received	FABIO	2017-05-17 11:39:40	192.168.1.88
logout	SVEN	2017-05-17 11:39:42	
logout	FABIO	2017-05-17 11:39:50	192.168.1.88

2.2 Interval Action Detector

The task of the *Interval Action Detector* is to assemble individual time-stamped action symbol with low-level labels into meaningful events described by an interval. In this way, each user session is modeled by means of a sequence of higher level intervals, rather than with a list of time-stamped low-level action symbols. More specifically, the *Interval Action Detector* aggregates the results from the OSN Crawler into medium-level events using medium-level predicates corresponding to the categories defined in [3] and shown in Table 2. Additionally, Table 3 shows the Interval Labeling obtained by processing the OSN Log shown in Table 1. In order to do that, we utilize interval-based extensions we have introduced in our earlier work on the detection of high-level events [5, 6]. All

Table 2. Facebook Predicates and related Action Symbols in Facebook logs

Category	**Atomic Action**
login	login
status&friends	status wall post friend approved mobile status update checkin status update
messages	message received message sent
photos	added picture tagged in a picture
shares	youtube video shared youtube created story link app created story published link link shared story video shared story pictured shared story
like	likes a page
logout	logout

operators, including the new ones for analyzing social network data, were implemented in PostgreSQL as stored procedures. We do not describe our extensions in detail, but introduce the concepts as needed (for details of our interval-based language, see [5, 6]).

The *Interval Action Detector* works both in an *offline* mode - where it has access to a complete (historical) data set - and in an *online* mode - where it works similar to a continuous query in a data stream management system, thus being able to react in real time. In addition, the GUI also allows to broaden both the category and the atomic action sets (Table 2), as well as to select just a subsets of the categories to be detected by the Interval Action Detector (by default, all of them are searched).

Table 3. Example of Interval Labeling

Pred	**Start**	**End**	Arg_1
session	17-05-17 11:39:12	17-05-17 11:39:50	FABIO
login	17-05-17 11:39:12	17-05-17 11:39:12	FABIO
like	17-05-17 11:39:13	17-05-17 11:39:20	FABIO
messages	17-05-17 11:39:21	17-05-17 11:39:40	FABIO
session	17-05-17 11:39:24	17-05-17 11:39:42	SVEN
login	17-05-17 11:39:24	17-05-17 11:39:24	SVEN
messages	17-05-17 11:39:25	17-05-17 11:39:27	SVEN
status&friends	17-05-17 11:39:28	17-05-17 11:39:30	SVEN
messages	17-05-17 11:39:31	17-05-17 11:39:40	SVEN
logout	17-05-17 11:39:42	17-05-17 11:39:42	SVEN
logout	17-05-17 11:39:50	17-05-17 11:39:50	FABIO

2.3 High-Level Event Detector

The main goal of the high-level event detector is to combine medium-level events into descriptions of complex events. This is done by putting the intervals associated with two or more medium-level events in relation to each other, thus exploiting the semantic interval relationships defined in [5] and used in *ISEQL* [6]. In principle, we have five different operators, here visualized by a small sketch indicating the relative position of two intervals: LEFT OVERLAP, DURING, START PRECEDING, END FOLLOWING, and BEFORE. All of these relations also have a reverse counterpart: RIGHT OVERLAP, REVERSE DURING, REVERSE START PRECEDING, REVERSE END FOLLOWING, and AFTER.

The constructs employed by ISEQL are not just simple Allen relationships, but we have extended and parameterized them to match the specific needs of event detection. For example, the relation BEFORE covers "takes place before" and "meets". The parameter δ of the BEFORE relation can be used to tune the desired maximum distance between the intervals, meaning that $\delta = 0$ models "meets", all other values model "takes place before".

We introduced two new constraints – the *cardinality* and *overlap percentage* [10], which allow us to formulate high-level event models in social network environments more easily, thus also enhancing the expressivity of ISEQL. These new operators were also integrated into the GUI.

For instance, we can utilize these new operators and constraints to define a *spamming* event model. We interpret spamming as sharing something more than k times *(Case 1, Fig. 3)* or spending more than p percent of a session with sharing activities *(Case 2, Fig. 4)*. In order to model and then detect such scenarios, we need to combine different intervals, each of them corresponding to a medium-level event. More specifically, for modeling *Case 1* we exploit the *right cardinality (k)* constraint [10] between "session" and "shares" intervals, which guarantees that there are at least k "shares" intervals within the same "session". On the other hand, for modeling *Case 2* we use the *left overlap percentage* constraint [10] between "session" and "shares" intervals, which states that at least p percent of the session was spent by the user *sharing* something. Fig. 3 and Fig. 4 show examples of instances that would be classified as spamming: Fig. 3 for $k = 3$ (Case 1), and Fig. 4 for $p = 0.8$ (Case 2).

While an experienced user can define high-level events directly in ISEQL, for an ordinary user this task may be too daunting. With the help of our GUI, a user is guided through the steps of defining a high-level event. This includes drawing intervals on a canvas and supplying parameters. In the background, the system checks the model for consistency, transforms it into a temporal relational algebra expression, and generates the code in form of *PL/pgSQL* for the actual event detection. As an example, Fig. 5 and Fig. 6 depict how the *spamming* high-level event (see Fig. 3 and Fig. 4) is entered using the GUI. More specifically, it is defined as logical disjunction of *Case 1* (Figure 3) and *Case 2* (Figure 4). Consequently, by answering some simple questions (Figure 5)[5], and drawing the

[5] We also provide an online help.

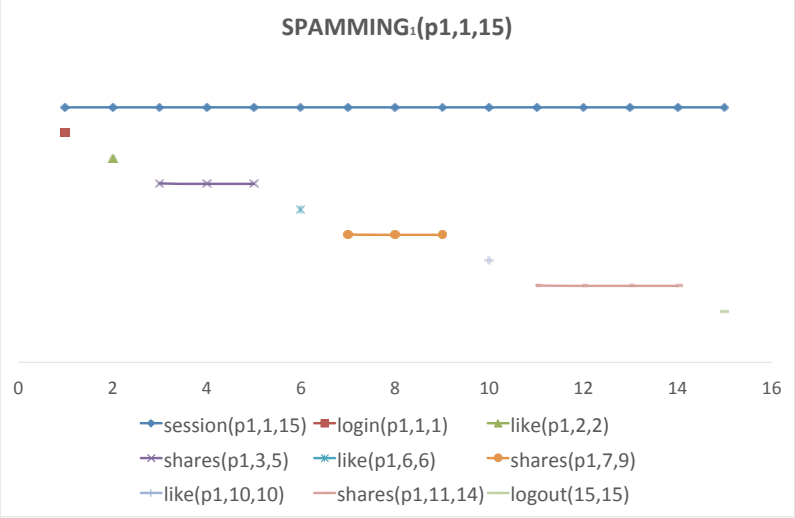

Fig. 3. Spamming - Case 1, $k = 3$

desired relationships among intervals on a temporal canvas (Figure 6), the user is able to effectively define a high-level event model. The black intervals in Figure 6 represent combinations of simpler events and are automatically generated by the system. This *high-level* event can now be stored, queried, and re-used as a building block to define more complex events. More details are provided in Appendix A.

By default, our queries work at a *global granularity*, looking at all sessions of all users. However, we can also run queries at a finer granularity, defining specific temporal intervals for users. We can even analyze a user's behavior by investigating and categorizing each of their sessions individually. The session type that appears most frequently is then used to classify a user. For example, a user most of whose sessions are marked as *inactive* - that means that they are not an instance of any of the searched event models - is classified as a *fake user*. In addition, we report some relevant screenshots classified by functionality, depicting how the overall system actually works[6].

Our framework is also flexible enough to enhance queries with data coming from Facebook graphs, adding further constraints to them. In this way, we can exploit the network structure of Facebook friendships to carry out the classification of *clusters of users* rather than single users. More specifically, we are interested in determining the category all the friends of a specific user u (i.e., his/her neighbors in the graphs) belong to, or extend the search to all users whose minimum distance from the user u is lower than or at most equal to a specific threshold d.

[6] https://www.dropbox.com/sh/um0yucb8810nrhu/AAAt5kbr9Tsz4moEgghKgxeja?dl=0

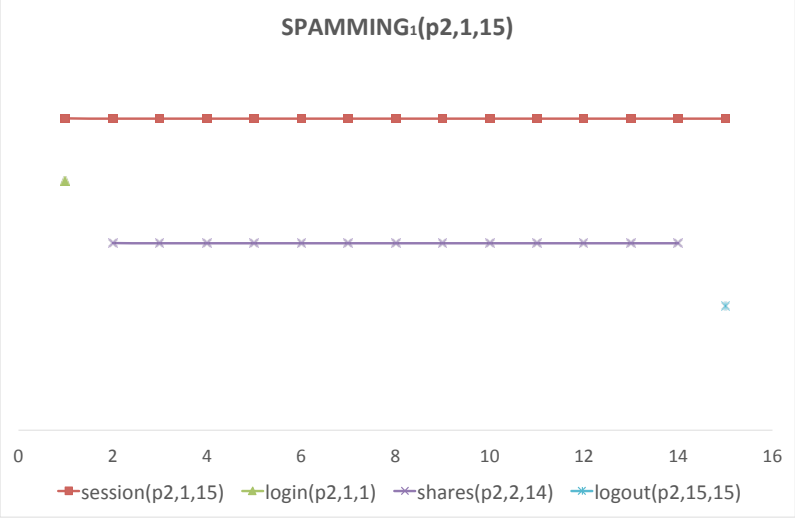

Fig. 4. Spamming - Case 2, $p = 0.8$

2.4 Graphical User Interface (GUI)

As shown in Fig. 2, each of the semantic layers of Fig. 1 consists of a *graphical user interface (GUI)*, an *application core*, and a *database*. Since many of the framework users may be inexperienced, both in social network analysis and in relational algebra, it may be non-trivial for them to define tasks and models. Thus, the GUI plays an important role in helping them to interact effectively with the framework and to make use of specific functionalities, such as the *definition of a new high-level event model*. More details about the functionalities of the framework and the way to utilize them via the graphical user interfaces are exhibited in Appendix A.

3 Demo Specifications

For the purpose of our demo, we use two different data sets: the Facebook data set we already mentioned earlier [1] and a synthetic data set produced by a generator. The size of the data set and the density of the events (i.e., the number of events per time unit) can be controlled via parameters by a user. We decided to use Facebook in our demo, since it is still the world's most popular social network; however, the *OSN Crawler* could potentially work also on other OSNs, such as Twitter, by using the Twitter APIs with the default access level.

We plan to start the demo by detecting high-level events on one of the above social data sets in offline mode to illustrate how the event detection works in principle. This involves identifying events such as different scenarios of potential spamming in the social log. However, depending on the particular interests of a

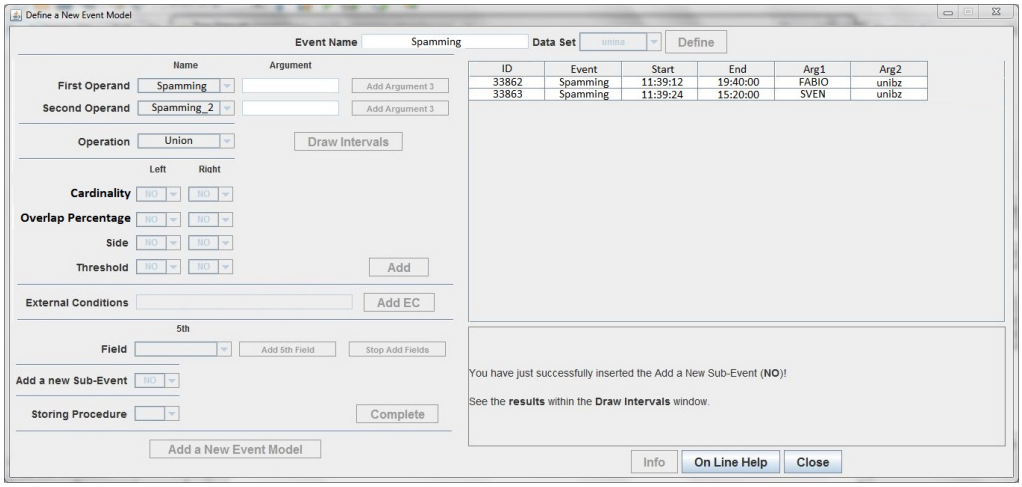

Fig. 5. High-Level Event Detector

demo participant we can either focus our attention on one specific layer (*OSN Crawler*, *Interval Action Detector*, and High-Level Event Detector) or carry out the whole process from the low-level label extraction all the way to the detection of high-level events. For the online mode, we stream one of the data sets past the event detector, emitting the atomic events according to their timestamps.

Participants will be able to discover some of the predefined medium-level events, investigate the interval labeling, or create new medium-level events and detect them afterwards. On the top-most layer, users can do the same for high-level events: detecting them and defining new ones.

All the functionality of every layer is accessible via intuitive GUIs that provide continuous feedback about what is happening in the system. For the interval action and high-level event detectors, a participant is not only able to define new events and detect them, but they can also have a closer look at the interval representation of events in the form of stored procedures, and define and detect them on the fly. They can also investigate the global behavior of a user over a specified temporal interval, thus obtaining the category of the user.

4 Conclusion and Future Work

In this paper, we present a smart and interactive framework for automatic event detection and user classification in a social network context. More specifically, the user is able both to easily define high-level event models by a means of a smart graphical user interface, and to discover their instances in a real data set containing data dealing with interactions of users with Facebook, as well as in synthetic data sets.

Future work will be devoted particularly to further enhance the framework efficiency. In fact, we plan to develop a family of efficient plane-sweeping interval

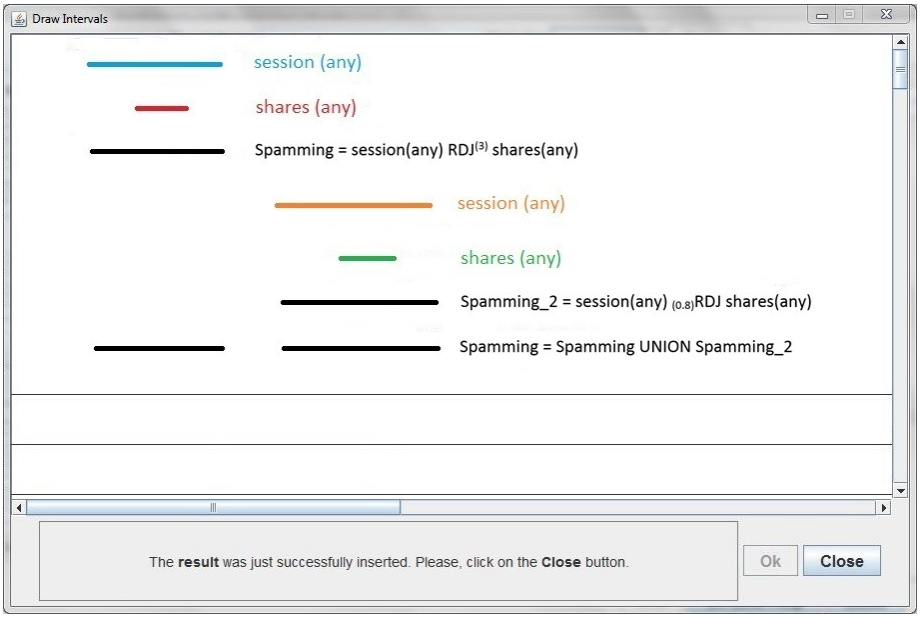

Fig. 6. High-Level Event Detector - Intervals on a Temporal Canvas

join algorithms that can evaluate the wide range of interval relationships predicates defined in [6], that are broadly exploited by our framework, directly in the query processing framework of the *PostgreSQL* DBMS. These predicates also include the *cardinality* and *overlap percentage* operators, that are particularly useful for modeling high-level events and user categories in a social network context. Additionally, we also plan to improve the framework response time for use in live data streams. This involves the development and the dissemination of a specifically designed and developed web application able to capture user interactions with a social network in real time, compatibly with its privacy policy.

Appendix A The Framework Functionalities

In this appendix we take a closer look at the main functionalities provided by the framework for high-level automatic event detection and user classification. More specifically, we list them below and give more details in the following sections.

- Detection of Low-Level Annotations (Section A.1).
- Detection of Medium-Level Annotations (Section A.2).
- Detection of High-Level Event Occurrences (Section A.3).
- Detection of User Classifications (Section A.4).
- Automatic High-Level Event Detection (Section A.5).
- Definition of a New Atomic Predicate (Section A.6).
- Definition of a New Medium-Level Predicate (Section A.7).
- Definition of a New High-Level Event Model (Section A.8).

A.1 Detection of Low-Level Annotations

This functionality allows to import all the low-level annotations occurring within a specified temporal window. So far, they can be imported from two different sources. These annotations are used as input for further processing steps. The first one is a real data set containing data related to the interactions of users with Facebook over a 2-year period, previously collected for [2]. The second one is a synthetic data set, generated by a specifically designated tool, whose size and the density of the events (i.e., the number of events per time unit) can be controlled via parameters by users. However, the system is flexible enough to easily allow in the future imports from other sources, including live data streams, compatibly with the related privacy policy.

A.2 Detection of Medium-Level Annotations

This functionality detects the interval labeling corresponding to the captured *OSN Log* (Fig. 1). More specifically, Fig. 7 shows the medium-level annotations corresponding to the OSN Log listed in Table 1. In Fig. 7 we use as *source* the synthetically generated data set *unibz* mentioned in Section A.1) and the medium-level predicates *status&friends*, *messages*, *photos*, *session*, *shares*, *like*, *logout* in *offline* mode. The collected interval labeling is shown on the right-hand side of Fig. 7 and can be further processed in order to infer both high-level events and user classifications.

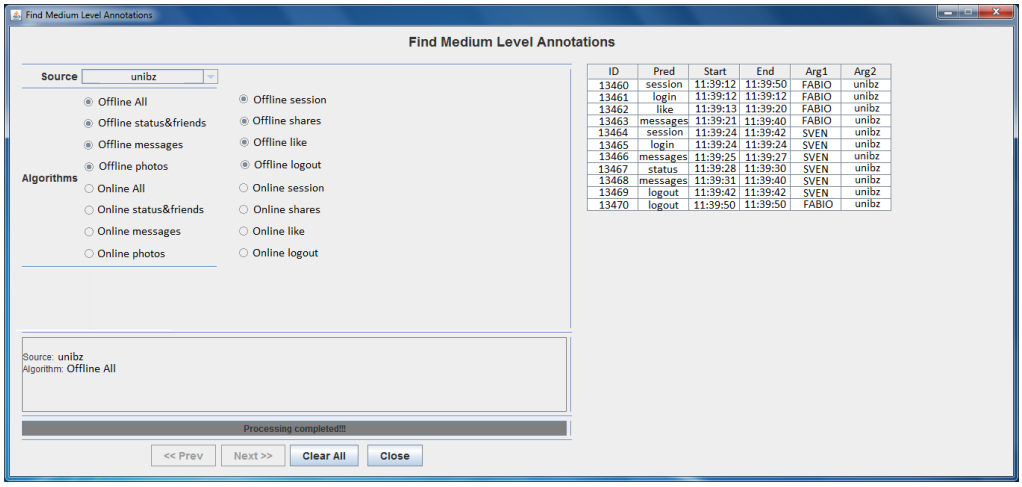

Fig. 7. Detection of Medium-Level Annotations

A.3 Detection of High-Level Event Occurrences

This functionality detects occurrences of high-level events whose models are stored in the knowledge base. In this framework the knowledge base of event models is stored as set of stored procedures in the *PostgreSQL* database management system. As shown in Fig. 8, the user simply needs to select the *event* to be discovered (*SPAM* in this case), and the data set to be investigated (*unibz* in this case). Clearly, for a data set to be available, it has to be first processed, i.e., it has to be labeled via *interval labeling*). The result of the use case shown in Fig. 8 is an instance of the *SPAM* event detected for the user *FABIO* from 11:39:12 to 19:40:00.

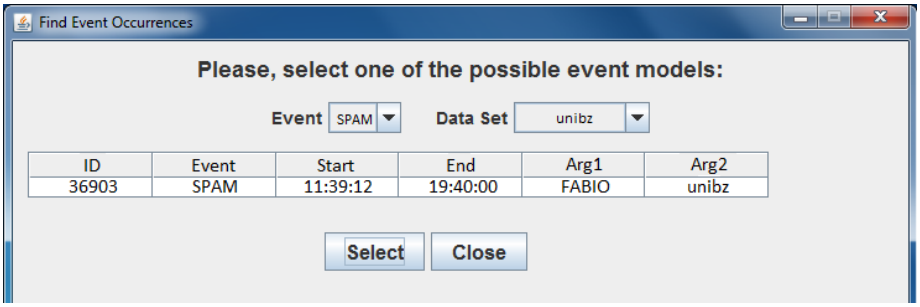

Fig. 8. Detection of High-Level Event Occurrences

A.4 Detection of User Classifications

Similarly to the procedure in Section A.3, this functionality allows us to discover the category to which each OSN user belongs. More specifically, a client interested in carrying out an OSN user classification has just to specify the following:

- the particular *OSN user* to be analyzed;
- the *time window* where he/she wants to classify the selected OSN user;
- the *data set* taken as reference.

As a result, a specific category is assigned to the OSN user depending on the classification of his/her sessions (spamming, status&friends, messages, photos, like, and inactive) that appears most frequently. Thus, the categories to which the OSN user could belong are respectively Spammer, Interactive with Friends, Message Sender, Photo Poster, Like Adder, and Fake User. This is due to the fact that all the defined event models are flexible, so they can be also applied to classify users themselves, thus working at a *lower (user) granularity*.

A.5 Automatic High-Level Event Detection

This functionality allows to automatically carry out the overall process described in Sections A.1, A.2, A.3, and A.4. As a result, the user just needs to specify all the inputs necessary in the previous sections once, and the whole process shown in Fig.1 is performed; consequently, the output are the high-level events and the user classifications satisfying the inserted constraints.

The process can be run in both offline and online modes. For the online mode, we stream one of the data sets past the event detector, emitting the atomic events according to their timestamps.

A.6 Definition of a New Atomic Predicate

This functionality allows the user to add another atomic event to the set of atomic actions listed in Table 2. For instance, the user in the use case shown in Fig. 9 inserts the atomic action named *Interact with Game*.

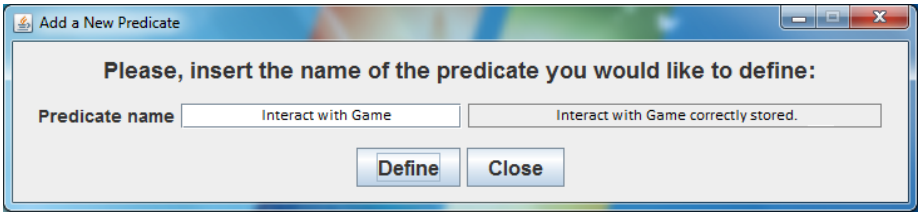

Fig. 9. Detection of High-Level Event Occurrences

A.7 Definition of a New Medium-Level Predicate

Similarly to adding atomic predicates as described in Section A.6, this functionality allows us to insert a new medium-level predicate into the set of categories listed in Table 2. More specifically, by means of another smart graphical user interface, the user is able to directly write the PL/pgSQL code of the new medium-level predicate, also specifying the relationships with the low-level atomic actions.

A.8 Definition of a New High-Level Event Model

As mentioned in Section 2, this functionality allows a user who is not familiar with relational algebra to easily define a high-level event model; Fig. 5 and Fig. 6 illustrate an example for using the smart graphical user interface for defining the *Spamming* event model.

In order to illustrate the advantages of the user interface, we describe the procedure for defining a *new event model* in the following. This is done in a step-by-step manner, by asking the user for (see Fig. 5):

- the *name* of the new event (field *Event Name*);
- the *data set* he or she would like to explore (from a list of available data sets) (field *Data Set*);
- the *medium-level predicates* (or, as an alternative, already-defined events) associated with the intervals (operands) that he or she is currently adding to the *global event* (fields *First Operand, Second Operand*);
- optional values for the *arguments* of the *first/second operand* in case of a medium-level predicate (field *Argument*, close to *First/Second Operand*); arguments can be easily added by clicking on the *Add Argument* button;
- the possibility to carry out set operations between the two inserted interval predicates (field *Operation*);
- drawing the two intervals (after clicking on the *Draw Intervals* button); then, the *application core* will capture the values of the *left* and *right endpoints of both intervals* (see for instance first and second lines of Fig. 6);
- specifying how often the left/right interval (fields *Left/Right Cardinality*, respectively) has to appear in the result set. If the user selects *YES*, a pop-up window will ask to select among three options; at least k times (k to be specified), more than one tuple ($*$), or exactly one tuple (*one*) [10]; otherwise, no further constraints are added;
- specifying the overlap percentage between the two intervals with respect to the left/right interval (fields *Left/Right Overlap Percentage*, respectively). In case the user selects *YES*, a pop-up window will ask for the overlap percentage (from 0% to 100%); otherwise, no further constraints are added;
- whether he or she wants to take into account the *relationships* between the left/right endpoints (fields *Left Side, Right Side*);
- the maximum distance between interval endpoints (fields *Left/Right Threshold*); in case of *overlapping events* checking whether to take into account the distance between the *left endpoints* of the *first* and *second operand* or between the *right endpoints* of the two *operands*. In case of *non-overlapping events*, a user has to specify whether to take into account the distance between the *right endpoint* of the *first operand* and the *left endpoint* of the *second operand* or between the *left endpoint* of the *first operand* and the *right endpoint* of the *second operand*. Depending on the information provided by the user, the application core infers the specific operator that will be applied.
- the optional *additional constraints* between the *first* and *second interval* he or she would like to add, starting from the *partial result set* (clicking on *Add EC*, close to the *External Conditions* field, and then allowing the addition of constraints via a mask);
- the *fields* he or she would like to project with reference to the current result set (field *Field*); the user just needs to select the fields to be projected, and click on *Add i-th Field*;
- whether he or she wants to add *more intervals* to the *complex event* he or she is defining (field *Add a new Sub-Event*); in that case, the process is repeated starting from the third bullet point;

– whether he or she wants to store the *event model* as a *PL/pgSQL procedure* (field *Storing Procedure*).

After each step the *application core* checks the consistency of the input. At the end of the procedure, a summary with the retrieved instances, if any, will be visible to the user.

References

1. Amato, F., Castiglione, A., De Santo, A., Moscato, V., Picariello, A., Persia, F., Sperli, G.: Recognizing human behaviours in online social networks. Computers & Security **74**, 355–370 (2018)
2. Amato, F., De Santo, A., Moscato, V., Persia, F., Picariello, A.: Detecting unexplained human behaviors in social networks. In: Proceedings - 2014 IEEE International Conference on Semantic Computing (ICSC 2014), pp. 143–150. IEEE, Newport Beach, CA, USA (2014)
3. Benevenuto, F., Rodrigues, T., Cha, M., Almeida, V.: Characterizing user navigation and interactions in online social networks. Information Sciences **195**, 1–24 (2012)
4. Dignös, A., Böhlen, M., Gamper, J.: Overlap interval partition join. In: International Conference on Management of Data, SIGMOD 2014, pp. 1459–1470. ACM, Snowbird, UT, USA (2014)
5. Helmer, S., Persia, F.: High-Level Surveillance Event Detection Using an Interval-Based Query Language. In: Proceedings - 2016 IEEE International Conference on Semantic Computing (ICSC 2016), pp. 39–46. IEEE, Laguna Hills, CA, USA (2016)
6. Helmer, S., Persia, F.: ISEQL: an Interval-based Surveillance Event Query Language. Int. J. Multimed. Data Eng. Manag. (IJMDEM) **7**(4), 1–21 (2016)
7. Irwin, A. S. M.: Double-Edged Sword: Dual-Purpose Cyber Security Methods. In: Cyber Weaponry: Issues and Implications of Digital Arms. pp. 101–112, Springer (2018)
8. Persia, F., Bettini, F., Helmer, S.: An Interactive Framework for Video Surveillance Event Detection and Modeling. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM 2017), pp. 2515–2518. ACM, Singapore, Singapore (2017)
9. Persia, F., Bettini, F., Helmer, S.: Labeling the Frames of a Video Stream with Interval Events. In: Proceedings - 2017 IEEE International Conference on Semantic Computing (ICSC 2017), pp. 204–211. IEEE, San Diego, CA, USA (2017)
10. Persia, F., Helmer, S.: A Framework for High-Level Event Detection in a Social Network Context Via an Extension of ISEQL. In: Proceedings - 2018 IEEE International Conference on Semantic Computing (ICSC 2018), pp. 140–147. IEEE, Laguna Hills, CA, USA (2018)
11. Piatov, D., Helmer, S., Dignös, A.: An interval join optimized for modern hardware. In: Proceedings - 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 1098–1109. IEEE, Helsinki, Finland (2016)
12. Schneider, F., Feldmann, A., Krishnamurthy, B., Willinger, W.: Understanding Online Social Network Usage from a Network Perspective. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, pp. 35–48. ACM, New York, NY, USA (2009)