User action representation and automated reasoning for the forensic analysis of mobile devices

User action representation and automated reasoning for the forensic analysis of mobile devices

COSIMO ANGLANO, University of Piemonte Orientale, Italy
MASSIMO CANONICO, University of Piemonte Orientale, Italy
LAURA GIORDANO, University of Piemonte Orientale, Italy
MARCO GUAZZONE, University of Piemonte Orientale, Italy
DANIELE THESEIDER DUPRÉ, University of Piemonte Orientale, Italy

We propose a framework for structuring the description and results of the forensic analysis of actions of investigative interest in digital applications, and for automated reasoning on such actions. A high level of abstraction is suitable for forensic stakeholders that are not ICT experts; other levels are suitable for automating experiments on the devices to establish traces left by actions, and for associating the results of the experiments. Such results are used in a computational logic framework to conclude evidence on the occurrence of actions. The evidence can be presented to stakeholders or used in further automated reasoning, and traced back to data on the device.

CCS Concepts: • **Computing methodologies** → **Knowledge representation and reasoning**; • **Security and privacy**;

Additional Key Words and Phrases: Digital forensics, Computational logic, Automated forensic analysis

ACM Reference Format:
Cosimo Anglano, Massimo Canonico, Laura Giordano, Marco Guazzone, and Daniele Theseider Dupré. 2021. User action representation and automated reasoning for the forensic analysis of mobile devices. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3465481.3470053

1 INTRODUCTION

The field of Digital Forensics (DF) has developed several methods and tools for the retrieval and analysis of evidence from digital devices. The field is however facing increasing challenges such as the dramatic increase in the demand for such a kind of analyses, which calls for more thorough methodologies, supported by adequate tools, and the requirement that the methods used in the analysis are verifiable and their results can be explained to actors (prosecutors, lawyers and judges) that are not ICT experts.

In order to support digital analysis of a device in a forensic case, digital evidence should be put in relation to action types performed by a user on the device, such as sending a message, possibly with multimedia attachments. Actions of the same type (i.e., all actions of sending a message with a given application) leave similar traces.

We use the term *a priori* analysis for the one that aims at identifying the relation between action types and their traces. A lot of work has been devoted, in particular, to the *a priori* analysis of messaging applications (see, e.g., [1, 3, 15]). The results of such *a priori* analysis can be used in a specific forensic case to search for instances of the traces which can be used as evidence for the occurrence of an action of the given type.

In the following we concentrate on Android devices. *AnForA* [2] is a tool that supports the *a priori* analysis of Android applications by automating a large part of the activities that have to be performed (see Section 2); this includes identifying sequences of elementary actions on the user interface that are ways for performing the action of interest.

The concrete actions to be performed on the user interface may be the same for different versions of an application, but the results of *a priori* analysis for an action type may be specific to some versions of an application and a given platform. In fact, different versions of the application on different operating systems may store information in a different way, e.g., in different files or directories (see, e.g., [2], section 5).

A repository can be organized to store the results of the *a priori* analyses for different combinations of action types, application version and platform version, and the work for the analysis for a given action type in a context (app and OS version) can be reused for analysing the same action type in a different one. AnForA supports this by carrying out a set of experiments in which actions of interest are automatically performed on the application under analysis by emulating a human user that interacts with its interface. Permanent storage of the device is monitored, so that the data created or modified by each one of the above actions can be correlated with that action. AnForA allows to repeat the same experiment for different versions of the application and/or of the mobile OS.

When traces on a specific device have to be analyzed in a forensic case, the results of the *a priori* analysis for an action type in the app+platform combination under examination will not necessarily be found in the repository. The DF expert could then use AnForA to perform in a supported way the *a priori* analysis for the app+platform versions under examination, in case a real or virtual device different from the seized one, which cannot be used for this purpose, with the same app+platform combination, is available for experimentation.

However, in a given context C (app+platform combination), traces may also be found that fit with the ones that are expected for an action type in a “similar” context C’ (e.g., same app version and different OS version; or a previous app version for the same OS).

Automated reasoning could be used to point out evidence for an action (an instance of an action type) both in case results of the *a priori* analysis are available for the context C under examination, and in case they are available for a “similar” context C’; the context C’ and its relation to the current one C would be pointed out, in order to make clear that the conclusion is valid under the assumption that the action-trace relation is the same (maybe only some of the expected traces are found, but this will be used as well). In fact, experience in the analysis of messaging applications suggests that it is extremely improbable (except, of course, in case the device has been tampered) that: traces that would be found in context C’ are present, the action-trace relation has changed from C’ to C, so that they are no longer evidence for the action, and they have been produced by something else.

The contribution of this paper is twofold. On the one hand, we propose a structured representation of user (macro)actions of interest at different levels of abstraction, in order to include:

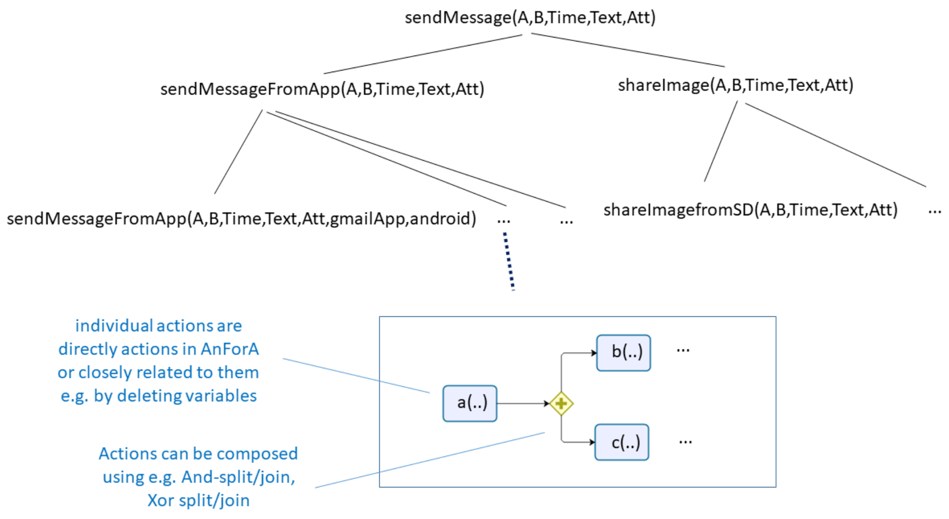
sendMessage(A,B,Time,Text,Att)
sendMessageFromApp(A,B,Time,Text,Att)
shareImage(A,B,Time,Text,Att)
sendMessageFromApp(A,B,Time,Text,Att,gmailApp,android)
...
...
shareImagefromSD(A,B,Time,Text,Att)
...
individual actions are directly actions in AnForA or closely related to them e.g. by deleting variables
a(..)
b(..)
c(..)
...
...
Actions can be composed using e.g. And-split/join, Xor split/join

Fig. 1. A sketch of the representation of actions for sending a message to a receiver with text and attachment(s).

(1) the level of interest for stakeholders (such as prosecutors), e.g.: A sent a message to B at time Time possibly with text Text and attachment Att[1];
(2) one that describes e.g. how a user can send a message to another one using a given application on a specific platform, at the granularity suitable for AnForA-supported *a priori* analysis, that is, performing instances of the action in AnForA;
(3) one that is specific enough for associating to the action of interest the precise traces that are expected to be found in permanent storage of the device (file system, SQLite databases of the applications).

On the other hand, we describe a computational logic model for reasoning on the action-traces relation, which also applies, as described before, to a similar context the results of *a priori* analysis for another context. The way actions are organized allows for using the results of reasoning at different levels of abstraction, in particular, for presenting them to the stakeholders, or using them for reasoning at a higher level, putting together different pieces of information corresponding to different actions of investigative interest.

In the next Section we provide a more detailed description of AnForA; in Section 3 we describe our proposal for structuring a representation of actions; we then describe in Section 4 the proposed computational logic approach.

2 ANFORA

AnForA [2] is a software tool that automates most of the activities that need to be carried out to forensically analyze Android applications. AnForA is based on a methodology for the forensic analysis of mobile applications that has been conceived in order to provide the following properties:

[1]In a comprehensive toolset, the contribution proposed in this paper can be complemented with natural language and image/video processing to conclude e.g. that “A sent B an intimidating message” or “A sent B an image of (illegal) category C”.

(1) *fidelity*, i.e. the ability to reproduce, as faithfully as possible, the interactions that a human experimenter would have with the application under analysis in order to perform the actions of investigative interest;
(2) *artifact coverage*, i.e. the ability to identify all the data that are generated/modified by the application as effect of the above actions;
(3) *artifact precision*, i.e. the ability to include only data generated/modified by the application under study;
(4) *effectiveness*, i.e. the ability to correlate each user action of interest with the data it modified/generated;
(5) *repeatability*, i.e. the ability to provide to a third party the possibility of replicating the same set of experiments and to obtain the same results;
(6) *generality*, i.e. the ability of generating results that hold for as many different devices as possible (possibly all).

The methodology providing the foundations of AnForA is based on the design of a set of experiments, each one focusing on one of the operations allowed by the application under analysis (e.g., sending a text message or a picture), on their systematic execution using the application on a mobile device, on the inspection of the device storage during and after each experiment (so as to identify the data generated during it), and on the analysis of the generated data to determine their meaning and context.

The methodology is articulated as follows:

(1) the functionalities of the application under consideration are analyzed, in order to identify its actions that have a potential investigative interest, and then suitable experiments, aiming at eliciting the generation of the data corresponding to the above actions, as well as their storage on the local memory of the device, are designed. Each experiment consists in a set of elementary actions (e.g., *TapXY(x,y)*, to tap at coordinate *(x,y)* of the device screen, and *SetTxt(id,txt)*, to insert text *txt* into the widget identified by *id*), that will be executed in a predefined order by interacting with the application user interface, so as to carry out an action of investigative interest;
(2) the application is installed on the device, and the set of directories that contain data generated either directly or indirectly by the application (the *Analysis Paths*) are identified, so that they can be monitored during the execution of the experiments in order to detect changes to the data they store;
(3) the set of experiments is carried out in a systematic way, until all of them are complete. In each experiment, the elementary actions are performed in sequence and:
    (a) a snapshot of the contents of the *Analysis Paths* is taken at the beginning of the experiment and after each elementary action;
    (b) at the end, the snapshots are:
        (i) compared in order to identify which files have been created, deleted and/or updated as effect of each elementary action;
        (ii) searched for known information (e.g., the text of a message that has been sent) to determine the data that have been written in, or deleted from, the above files.

3 ACTION REPRESENTATION

The actions to be considered for the analysis with AnForA of different applications of the same class, such as messaging applications, have something in common. We propose to organize knowledge about them as an ontology of (macro)actions, with abstractions along several dimensions, e.g.:

- ignoring variables (as in figure 1, "sending a message with the Gmail app on Android", and other parameters, is a subclass of "sending a message" - and the same parameters);
- use of generic parameters with respect to using specific ones, e.g., the Gmail app vs Gmail app version 8.5.6; note that this may also be seen as ignoring variables that provide the app/OS version;
- distinguishing parts of the macro action that are performed in different ways on the device (e.g., again referring to figure 1, sharing an image from the SD card, from internal memory, or a photo to be shot with the camera).

A description of macro actions in terms of individual actions (close to elementary actions in AnForA) should be provided at the highest level where this is appropriate, in the sense that such description is valid for all more specific actions. With respect to figure 1, the individual actions for sending a message with different versions of the same app could be the same for all versions. The action for sharing an image from the SD card may have, as we shall see, a common part for selecting the image, and different parts depending on the app that is selected; it may be partly different from the action for sharing a photo that is produced with the camera during the action itself. Macro actions are described as follows:

- the basis is given by individual actions (a, b, c in Figure 1) that are elementary actions in AnForA, or closely related to them: with an individual action, instances in terms of elementary AnForA action, with the additional information for being actually executed, should be associated (see section 3.1 for an example);
- individual actions are composed in a sequence or workflow; in this paper, in particular, we assume that a structured workflow is described, using sequence, and-split/join, xor-split/join.

In the lower part of figure 1, a macro action is described by the action a with some parameters, followed by an and-split of two sequences of actions, starting with b and c respectively (we use the BPMN notation [17] even though it was developed for a different purpose).

Possible executions of a macro action in terms of AnForA elementary actions can be derived by unfolding the workflow, in the example:

$a(\ldots), b(\ldots), c(\ldots), \ldots$

$a(\ldots), c(\ldots), b(\ldots), \ldots$

and replacing individual actions with AnForA elementary actions associated with them.

In the artifacts we expect to retrieve (some of) the parameters of the macro action and individual actions, such as the name of the file being sent in attachment.

3.1 Example action model

Figure 2 is an example model for the action of sharing an image from the SD card of a smartphone. Some actions are close to elementary actions in AnForA, but they are still an abstraction of them. For example, TapXY(sdCardRect) represents a tap on the rectangle of the screen which implies selecting the SD card. The actual position of the rectangle "sdCardRect" can be provided separately, depending on the screen size,

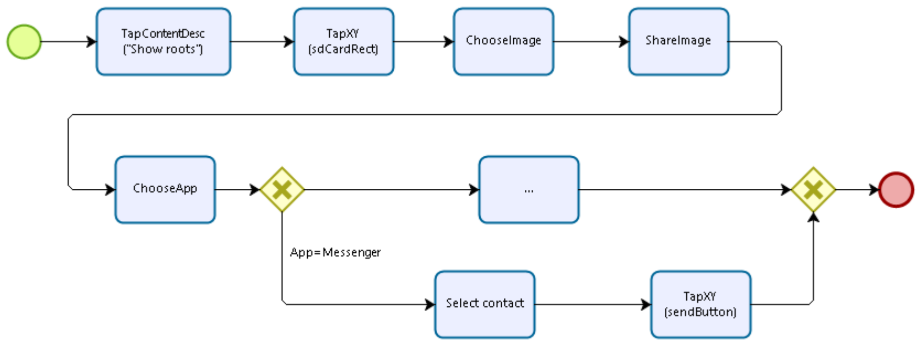

Fig. 2. Example model for sharing an image from the SD card.

etc; an instance (an elementary action in AnForA) of the action is a TapXY in a point within the rectangle. This abstraction is intended to be useful because the actual position of the rectangle and the actual point of the tap action (provided it is within the rectangle) are assumed not to make a difference in the traces which are generated by the action - in this case, actually, the tap action itself does not leave traces.

3.2 Traces for the example

The analysis of the example macro action with AnForA [2] for the case where the selected application is Facebook Messenger version 237.0.0.11.108 reveals that the following relevant artifacts are produced:

- A file (the one that is actually sent, generated by Messenger from the file selected by the user) is in a temporary directory specific to Messenger. It is worth noting that such file does not necessarily persist for a long time on the device, since, being stored in a temporary directory, it will eventually be removed to save space. Therefore, in general, this file might not be available for the analysis; however, modeling this would require more effort either in analyzing the behaviour of the application over time, or analyzing its code.
- In the "messages" table of the "threads_db2" database (which is an SQLite database where Messenger stores details about the conversations exchanged) a tuple is inserted for each message exchanged, with:
  - a "timestamp_ms" attribute (7th column), representing the timestamp the message was sent or received;
  - in the "sender" (5th column) attribute, a Json object, with the Facebook ID of the sender associated with the key "user_key";
  - a "thread_key" attribute (3rd column) of the form 'ONE_TO_ONE:receiverID:senderID' (where "receiverID" and "senderID" are the Facebook IDs of the sender and the receiver, respectively);
  - a "text" attribute (4th column), containing the text exchanged in the message;
  - an "attachments" attribute (9th column), a Json element with the name of the sent file associated with the "filename" key.

- In the "thread_participants" table of "threads_db2" a tuple is inserted/updated for each conversation thread (i.e., group of related messages) with information related to participants involved in the conversations (e.g., their Facebook ID and the last read receipt timestamp).
- Furthermore, the table "thread_users" of "threads_db2" contains information from Facebook on the users corresponding to the IDs found in the above tables.

4 AUTOMATED REASONING ON USER ACTIONS

In this section we describe how automated reasoning on actions can be performed in computational logic (CL) frameworks. The goal is to establish whether on the device being analyzed there is evidence of the occurrence of actions that have been the subject of *a priori* analysis. Different CL frameworks could be used to the purpose, but in the following we consider Answer Set Programming (ASP) [12].

The goal above requires representing in the CL model at least the following:

A) Information on which applications and versions are installed on the device, and on the applications and contexts for which the results of *a priori* analyses are available.

B) Knowledge resulting from *a priori* analyses, relating relevant user actions to the artifacts they produce, for the applications being considered in the analysis. In the following we do not consider the case of artifacts that remain after an application has been uninstalled. Then we only consider traces relative to an installed application (using results of *a priori* analyses for close versions).

C) Information from permanent storage of the device, relevant to establish the presence of the types of artifacts considered, such as: information on the contents of (relevant parts of) the file system; relevant information from the SQLite databases of the applications.

Before going into more details, let us discuss how reasoning could be performed about evidence that actions have occurred. In a general setting, establishing that an action has occurred would require abductive reasoning about actions and change in a temporal context (as, e.g., in [10]), that is, given:

- knowledge of the effects of actions, in the form "if action then traces", or formulas that imply this;
- knowledge about whether such effects persist (including knowledge about actions that may remove them);
- information on effects present at current time;

the occurrence of actions at previous times should be hypothesized, so that the hypothesized (sequence of) actions explains (implies) the presence of the effects.

As seen in section 3, relevant actions considered in this paper leave more than one artifact. In general, one could expect that:

- Some effects of relevant actions may disappear. As we have seen, this indeed occurs at least for temporary files that are generated by relevant actions and are removed later to save space. However, it may be difficult to model when such removal actions occur (so that they may be hypothesized only in some cases); it might be the case that the application periodically clears the directory, but establishing when this happens requires a different type of *a priori* analysis of the behavior of the application; or an application for freeing up space might have been run - and such an action may not leave traces itself.

- A given artifact might have been generated by different actions. E.g., consider a case where action a produces effects e_1 and e_2, action b produces e_1 and c removes e_2. The presence of e_1 without e_2 could then be explained either by the occurrence of a and then c, or by the occurrence of b. Consider a different case where: a produces effects e_1 and e_2, action b produces e_1 and c produces e_2. The presence of e_1 and e_2 could then be explained either by the occurrence of a, or by the occurrence of b and c. In practice, however, the artifacts produced are typically specific to individual applications, i.e., — again, except in case of tampering to produce false evidence — it is not the case that different actions (at an appropriate level of the abstraction hierarchy in figure 1) produce a same artifact; or, for cases where this happens (e.g., different applications adding an entry to the contacts) they also produce artifacts that are specific to the action.

Given the considerations above, we provide a solution as follows. As discussed above, in a declarative model of the way the system works, the action-evidence relation should be expressed in logic using rules of the form "if action then evidence". However, assuming that knowledge on possible causes for effects is complete, abduction can be reduced to deduction [8]: in general, the disjunction of possible causes for an effect is deduced from it. In this case, there is, essentially, a single possible cause for each combination of artifacts that can be produced by a relevant action, so no disjunction is necessary.

Rules to conclude that there is *evidence* for an action can then be provided as follows. In some cases, it is possible to distinguish pieces of evidence that can be considered permanent from the ones that are considered volatile (like temporary files). In such cases, rules are given for concluding that there is evidence for the action in case all permanent pieces of evidence are present. When such a distinction is less clear, rules are provided to conclude there is at least one piece of evidence for the action, i.e., one of the expected artifacts (given that one or more might have been canceled). An alternative is to provide rules to conclude *partial evidence* for an action in case there is at least one piece of evidence, and *complete evidence* in case all of them are there.

In all such variants, rules in part B of the model serve both the purpose of representing knowledge about the results of *a priori* analyses, and reasoning to conclude that actions have presumably occurred. This is not ideal from a Knowledge Representation and Reasoning point of view, where a model of a domain or system should in principle be expressed independently of the forms of reasoning applied to it — including deductive and abductive reasoning for, e.g., prediction, diagnosis, explanation, planning, plan recognition. However, the representation adopted in this paper provides what is strictly necessary for the purpose of establishing evidence for actions, and we have provided justifications for it.

We then sketch the representation. Part C means representing in ASP part of the information in the file system and the application databases. Detailing this does require some effort, but this is similar to what should be done by any tool using the results of *a priori* analysis for a specific device: in the example in Section 3, the Facebook IDs of sender and receiver should in any case be extracted from the string containing them.

For part A, in order to simplify the description, in the following we limit to consider different major numbers of application versions, not different versions of the operating system, given that differences in the results of *a priori* analyses tend to occur mainly with major changes of the application.

The fact that an application is installed on the device under examination is represented with facts of type:

```
use(Appname,Thisversion):-
     thiscontext(Appname,Thisversion),
     availableAPA(Appname,Thisversion).
use(Appname,Version):-
     thiscontext(Appname,Thisversion),
     not availableAPA(Appname,Thisversion),
     close(Appname,Thisversion,Version).
close(Appname,Thisversion,Version):-
     thiscontext(Appname,Thisversion),
     availableAPA(Appname,Version),
     not avAPAinbtw(Appname,Thisversion,Version).
avAPAinbtw(Appname,Thisversion,Version):-
     thiscontext(Appname,Thisversion),
     availableAPA(Appname,Version),
     availableAPA(Appname,V1),
     Version<V1,V1<Thisversion.
% and a similar rule for the case
% Thisversion<V1, V1<Version.
```

Fig. 3. Rules for selecting APA to be used.

```
evidence(shareImagefromSDwithApp(fbmessenger,Sender,Receiver,T,Text,Filename)):-
     use(fbmessenger,237), % this rule is derived from APA for Messenger version 237
                          % and if "use(..)" is true, the rule can be used
     % ... details on how to get the information from artifacts
     messenger_threads_db2_messages(...),
     messenger_threads_db2_thread_users(...),
     ...
evidence(sendMessage(Sender,Receiver,T,Text,Filename)):-
     evidence(shareImage(Sender,Receiver,T,Text,Filename)).
evidence(shareImage(Sender,Receiver,T,Text,Filename)):-
     evidence(shareImagefromSD(Sender,Receiver,T,Text,Filename)).
evidence(shareImagefromSD(Sender,Receiver,T,Text,Filename)):-
     evidence(shareImagefromSDwithApp(fbmessenger,Sender,Receiver,T,Text,Filename)).
```

Fig. 4. Rules for inferring evidence of actions. Here, messenger_threads_db2_messages() is a representation in ASP of the messages() table in the SQLite database threads_db2 of the Messenger application; similarly for ...thread_users().

```
thiscontext(appname,majorVersionNumber)
```

while the fact that results for *a priori* analyses (APA) are available for a version of an application is given as:

```
availableAPA(appname,majorVersionNumber)
```

so that, for example, we may have:

```
thiscontext(fbmessenger,240).
availableAPA(fbmessenger,237).
```

In part B, knowledge to be used for the analysis of an installed application can be determined as in Figure 3. In this way, if knowledge is available for the installed version, that one will be used; otherwise, if it is available for smaller/higher versions, knowledge for the closest ones in either direction can be used. In the example above, where version number is 240, and APA is only available for version 237, `use(fbmessenger,237)` will be inferred, meaning: try and use APA about version 237.

Then, evidence for an action relative to an application can be inferred with rules of the form:

```
evidence(actionname(appname,...)):-
    use(appname,version),
    % details on how to get the information
    % from artifacts based on APA
    % for appname and version
```

As discussed before, the premise of the rule should detect the combination of "permanent" artifacts for the action, but other variants are possible.

The first rule in figure 4 is the first part of the rule for the example in Section 3. The part relative to the results of APA depends on part C of the representation.

Additional rules of the form:

```
evidence(A1):- evidence(A2).
```

like the further rules in figure 4, are provided for all pairs A1, A2 of action types such that, in the ontology of actions (as in figure 1), A2 is (directly) more specific than A1 (this is a way for representing this in ASP).

Given the representation above, using rules deriving from *a priori* analyses of various specific actions in different contexts, instances of a generic action, like `sendMessage(A,B,Time,Text,Filename)` can be derived. The rules and data used for deriving a specific instance can be traced in order to provide support for the evidence, where needed; on the other hand, the conclusions relative to the abstract action occurrences can be used for presenting information to stakeholders. Such pieces of information, which abstract from the specific way an action has been performed on the device, are also suitable, in perspective, to be used in higher level automated reasoning, where evidence for single actions of investigative interest should be put together to support or suggest investigative hypotheses.

5 CONCLUSIONS AND RELATED WORK

In this paper we presented the basic ideas of a framework where actions of interest for the forensic analysis of digital devices are structured at different levels of abstraction, including: one that is useful for stakeholders that are not ICT experts, one suitable for performing experiments on the devices to establish traces left by actions of investigative interest, and one suitable for associating the results of the experiments. Reasoning in computational logic is used, based on such results and on the structured representation of actions, to conclude evidence on the occurrence of actions. Such evidence can be presented to stakeholders or used in further automated reasoning, and traced back to data on the device.

Existing state-of-the-art tools (e.g., [6, 16, 18]) currently support only the extraction and decoding of the information generated by applications and stored on a digital device, that is they are not able to determine the user action starting from stored data. AnForA provides the basis for the development of digital forensic analysis platforms with the above capability for Android devices, and the proposal in this paper is a step in this direction.

Modeling events at different levels of abstraction for DF analysis was proposed already by Carrier and Spafford [4].

The use of computational logic in digital forensics was pioneered by Costantini et al. [9], using ASP for its capabilities in solving combinatorial problems; in this paper we address a different problem, but the proposed model can be part of a larger automated reasoning system exploiting ASP.

A recent survey [11] summarized the state of the art in the application in Digital Forensics of Artificial Intelligence, mainly intended as Machine Learning and Deep Learning; some exceptions for event reconstruction are described in the following and are related to the approach in this paper which is based on Symbolic AI and Computational Logic in particular.

Hargreaves and Patterson [14] proposed an approach for timeline reconstruction based on high-level events (e.g. Google searches, connecting a USB device) that are described in terms of low-level events; in particular, a triggering low-level event and other low-level events are associated with the high-level one, and if the triggering event is present, the other low-level events of given types are searched with a timestamp difference within a given amount.

Chabot et al. [7] proposed an ontology for representing digital events of interest for an investigation. A knowledge graph based on the ontology is populated with information extracted by *log2timeline* [13]. Special-purpose inference is used to conclude that events temporally close to other ones are attributed to the same user, and to provide a correlation score to events based on their temporal correlation and their relation to same subjects and objects.

Turnbull and Randhava [19] propose describing digital events at different levels of abstraction so that, in particular, the presence of higher level events can be inferred. The approach is based on RDF representation and ad-hoc reasoners.

The proposal in this paper is oriented at being integrated with [2] and using a general Computational Logic reasoner; future work could possibly exploit the representations proposed in the work mentioned above as well as the one on CASE [5], which is explicitly developed to support tool interoperability.

ACKNOWLEDGMENTS

The work in this paper was partly supported by Università del Piemonte Orientale and by INDAM-GNCS Project 2020. It was developed in the context of the European Cooperation in Science & Technology (COST) Action CA17124 “Digital Forensics: Evidence Analysis via Intelligent Systems and Practices”.

REFERENCES

[1] Cosimo Anglano, Massimo Canonico, and Marco Guazzone. 2017. Forensic analysis of Telegram Messenger on Android smartphones. *Digit. Investig.* 23 (2017), 31–49.

[2] Cosimo Anglano, Massimo Canonico, and Marco Guazzone. 2020. The Android Forensics Automator (AnForA): A tool for the Automated Forensic Analysis of Android Applications. *Comput. Secur.* 88 (2020).

[3] Konstantia Barmpatsalou, Tiago Cruz, Edmundo Monteiro, and Paulo Simoes. 2018. Current and Future Trends in Mobile Device Forensics: A Survey. *ACM Comput. Surv.* 51, 3 (May 2018).

[4] Brian D. Carrier and Eugene H. Spafford. 2006. Categories of digital investigation analysis techniques based on the computer history model. *Digit. Investig.* 3, Supplement-1 (2006), 121–130.

[5] Eoghan Casey, Sean Barnum, Ryan Griffith, Jonathan Snyder, Harm M. A. van Beek, and Alex Nelson. 2017. Advancing coordinated cyber-investigations and tool interoperability using a community developed specification language. *Digit. Investig.* 22 (2017), 14–45.

[6] Cellebrite LTD. [n.d.]. Cellebrite UFED. https://www.cellebrite.com/en/ufed.

[7] Yoan Chabot, Aurélie Bertaux, Christophe Nicolle, and M. Tahar Kechadi. 2015. An ontology-based approach for the reconstruction and analysis of digital incidents timelines. *Digit. Investig.* 15 (2015), 83–100.

[8] L. Console, D. Theseider Dupré, and P. Torasso. 1991. On the Relationship between Abduction and Deduction. *Journal of Logic and Computation* 1, 5 (1991), 661–690.

[9] Stefania Costantini, Giuseppe De Gasperis, and Raffaele Olivieri. 2019. Digital forensics and investigations meet artificial intelligence. *Ann. Math. Artif. Intell.* 86, 1-3 (2019), 193–229.

[10] Marc Denecker, Lode Missiaen, and Maurice Bruynooghe. 1992. Temporal Reasoning with Abductive Event Calculus. In *Proc. ECAI 92*. 384–388.

[11] Xiaoyu Du, Chris Hargreaves, John Sheppard, Felix Anda, Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. 2020. SoK: Exploring the State of the Art and the Future Potential of Artificial Intelligence in Digital Forensic Investigation. In *Proceedings of the 15th International Conference on Availability, Reliability and Security* (Virtual Event, Ireland) *(ARES '20)*. Association for Computing Machinery, New York, NY, USA.

[12] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. 2012. *Answer Set Solving in Practice*. Morgan & Claypool Publishers.

[13] Kristinn Guðjónsson. 2010. Mastering the super timeline with log2timeline. https://www.sans.org/reading-room/whitepapers/logging/paper/33438.

[14] Christopher Hargreaves and Jonathan Patterson. 2012. An automated timeline reconstruction approach for digital forensic investigations. *Digit. Investig.* 9 (2012), S69–S79.

[15] Giyoon Kim, Soram Kim, Myungseo Park, Younjai Park, Insoo Lee, and Jongsung Kim. 2021. Forensic analysis of instant messaging apps: Decrypting Wickr and private text messaging data. *Forensic Science International: Digital Investigation* 37 (2021), 301138.

[16] MSAB. [n.d.]. XRY product family. https://www.msab.com/products/xry.

[17] Object Management Group. [n.d.]. Business Process Model and Notation . https://www.bpmn.org/.

[18] Oxygen Forensic. [n.d.]. Oxygen Forensic Detective. https://www.oxygen-forensic.com/en/.

[19] Benjamin Turnbull and Suneel Randhawa. 2015. Automated event and social network extraction from digital evidence sources with ontological mapping. *Digit. Investig.* 13 (2015), 94–106.