Engineering User-centered Explanations to Query Answers in Ontology-driven Socio-technical Systems

Juan Carlos L. Teze a, Jose Nicolas Paredes b, Maria Vanina Martinez c and Gerardo Ignacio Simari b,*

a Departamento de Ciencias e Ingenieria de la Computacion & Instituto de Ciencias e Ingenieria de la Computacion (UNS–CONICET), Universidad Nacional del Sur (UNS), Argentina
E-mails: jose.paredes@cs.uns.edu.ar, gis@cs.uns.edu.ar

b Facultad de Ciencias de la Administracion & Consejo Nacional de Investigaciones Cientificas y Tecnicas (CONICET), Universidad Nacional de Entre Rios (UNER), Argentina
E-mail: carlos.teze@uner.edu.ar

c Departamento de Computacion, Universidad de Buenos Aires (UBA) & Instituto de Ciencias de la Computacion (ICC UBA–CONICET), Argentina
E-mail: mvmartinez@dc.uba.ar

Abstract. The role of explanations in intelligent systems has in the last few years entered the spotlight as AI-based solutions appear in an ever-growing set of applications. Though data-driven (or machine learning) techniques are often used as examples of how opaque (also called black box) approaches can lead to problems such as bias and general lack of explainability and interpretability, in reality these features are difficult to tame in general, even for approaches that are based on tools typically considered to be more amenable, like knowledge-based formalisms. In this paper, we continue a line of research and development towards building tools that facilitate the implementation of intelligent socio-technical systems, focusing on features that users can leverage to build explanations to their queries. In particular, we propose user-centered mechanisms for building explanations based both on the kinds of explanations required (such as counterfactual, contextual, etc.) and the inputs used for building them (such as the ABox, TBox, and lower-level data-driven modules). In order to validate our approach, we develop a use case in the domain of hate speech in social platforms, using a recently-proposed application framework, and make available the source code for this tool1.

Keywords: Ontological Languages, Socio-Technical Systems, Explainable Artificial Intelligence, Hate Speech in Social Platforms

1. Introduction

Sharing data through a wide range of social platforms has now become the norm—such platforms are an example of one particular class of socio-technical system [1]. Even though these systems are very useful tools to connect people socially (as well as in other settings like commerce and work, among others), they are also notoriously vulnerable—it is now commonplace to see attacks taking many forms, such as cyberbullying, hate speech, fake

---

*Corresponding author. E-mail: gis@cs.uns.edu.ar.

1 https://github.com/jnparedes/HEIST

1570-0844/$35.00 © 2022 – IOS Press. All rights reserved.
Contributions. The contributions of this work can be summarized as follows:
(i) We develop an instantiation of the HEIC (Hybrid Explainable and Interpretable Cybersecurity) Application Framework\(^2\) (recently proposed in [10, 11], here generalized to other domains and renamed to HEIST – Hybrid Explainable and Intrepretable Socio-Technical systems), which yields the implementation of a system—called NetDER-HS—for detecting hate speech in online social platforms.

(ii) We explore and analyze seven different kinds of primitive explanations in the context of AI systems that leverage ontological knowledge and network diffusion-driven simulation tools.

(iii) We characterize the explanations presented above in the context of the HEIST framework, and show how they can be applied in the NetDER-HS system.

(iv) We develop a “build-your-own”-style approach for users to choose explanations according to their requirements. Furthermore, we analyze the 21 possible ways in which the primitive explanation styles can be combined in pairs, discussing the interactions and constraints that arise in these combined styles.

(v) We make available a preliminary version of the implementation of the HEIST application framework and document its use, focusing on the design aspects necessary to build explanations following the basic styles studied in this work.

The article is structured as follows: Section 2 provides a brief overview of various issues related to explainability in socio-technical systems, focusing on cybersecurity as an example domain, and introduce the HEIST framework for XAI-based socio-technical systems; Section 3 develops a use case that arises from the instantiation of HEIST; in Section 4, we explore how different kinds of explanations in the context of HEIST can be built; Section 5 provides further details by documenting the design and implementation of explainable systems based on our framework; and finally, Sections 6 and 7 describe the related work and conclusions, respectively.

2. Background on Explainability in Intelligent Socio-technical Systems

Explainability has been identified as a key factor for adoption of AI in a wide range of contexts, as the increasing deployment of intelligent systems in different domains has shown that when decisions are taken by these systems, it is essential for practical and social reasons that explanations are available to users. There has been an explosion in the number of works in the literature exploring XAI; in order to have the possibility to provide an adequate background, here we will focus on XAI in cybersecurity, which can be seen as a sub-domain of social-technical systems. We begin with the work of [6], which discusses several questions that should receive special attention to tackle the challenges of XAI-based cybersecurity. Then, we present the kinds of explanations that we later contemplate in our proposal. Finally, we briefly review the application framework for XAI-based cybersecurity systems we recently proposed in [10]; here, we generalize it to other domains and rename it to HEIST – Hybrid Explainable and Intrepretable Socio-Technical systems.

2.1. Explainable Cybersecurity: Questions Guiding R&D

In [6], the authors highlight several research directions by describing different aspects related to explanations in the cybersecurity context such as who gives/receives the explanations, what constitutes a good explanation, when the explanation should be given, how to give explanations, where the explanations should be made available, and why the explanations are needed. We now briefly discuss each of these questions.

Who? – As we have mentioned, cybersecurity is closely related with socio-technical systems because it combines human efforts with those of more traditional information systems. On the human side, participants occupy roles that range from novice, barely computer-literate users to experts who can write their own code. In particular, malicious actors also fall somewhere in this range, since nowadays there are tools that allow to launch attacks with very little knowledge. Thus, the level of expertise of the user plays a important role in defining an explanation and how explicit it should be.

\(^2\)In software engineering, application frameworks are general-purpose structures (such as abstract architectures) designed to support the production of domain-specific applications that are created by instantiating (or extending) the framework.
What? – Another key related question is to identify the object to be explained; in this context, the level of detail will depend on the user’s goals. For example, novice users will likely need explanations that improve their trust in system answers, or that help them to understand how to use the system correctly, whereas further details will be necessary for power users. So, different components will need to be explained, such as automated mechanisms, policies, threat models, vulnerabilities, and countermeasures.

Where? – Location is another issue that needs to be addressed in order to decide where it is best for explanations to be made available. Possibilities include different levels, as part of specific workflows, detached from system functionalities, or incorporating dedicated modules to handle features related to explanations.

When? – Explanations can be provided at various instances—before, during, or after a task is carried out. Explainability may be essential for the system to begin functioning (for instance, to explain aspects of input data), it may be crucial to support users’ trust at runtime, or the goal may be to support future decisions, making it important for explanations to be available after the execution.

Why? – One of the most important questions regarding explainability is the reason why we need an explanation, since it points to the importance of the users’ understanding of the way the system produces its outputs. Some examples include increasing confidence, trust, transparency, usability, verifiability, or testability. Moreover, the importance and need for explanations generally fall into two classes: critical and beneficial. In the former, the system’s outputs will not be accepted without adequate explanation, while in the latter such explanations are beneficial, but not crucial.

How? – Finally, a delivery style suitable for the intended audience must be chosen. Styles can be categorized according to language, including:

– **natural language** to produce possibly informal text or sentences that follow a template-based approach;
– **graphical language** to leverage the intuitions of visual components such as explanation trees, graphs, message-sequence charts or statistical histograms;
– **formal language** including code, proofs, and the system’s internal representations; and
– **games** to teach users how to interact with the system.

2.2. Explanations in Intelligent Socio-Technical Systems

As we began to argue in the previous section, a key requirement for the success and practical adoption of Intelligent Socio-Technical Systems (ISTSs, for short) is that users must have confidence in the system’s outputs and automated decisions. In this sense, automatically generated explanations have been recognized as a fundamental mechanism to increase user trust in AI applications. With this motivation, here we will analyze several kinds of explanations that will be used in the rest of the paper. For a more comprehensive review of the literature on XAI, we refer the reader to [12–14].

Contrastive: Refers to approaches where explanations are based on a strategy that focuses on establishing some type of contrast [13, 15, 16]—i.e., seeking to answer the question “why was one decision made instead of another”. The contrast is then made between the resulting decision and one or more alternative decisions of interest. The goal of this approach is to provide users with sufficiently comprehensive information on the reasoning the led to the system’s outputs by answering questions focusing on possible alternatives that may be more intuitive than the ones received.

Counterfactual: As a sub-class of contrastive, counterfactual explanations identify the required changes on the input side that would have significant impact on the output [16, 17]. People are used to considering counterfactuals in daily life; for example, adding new information to incomplete knowledge, making assumptions to evaluate possible scenarios, etc. Counterfactuals have become relevant to a variety of AI applications (e.g., improve training in generative adversarial networks (GANs)), and especially in XAI; the strategy has been particularly successful in systems that do not rely on data-driven techniques [18]. Although this explanation type is often seen as human-friendly, one drawback that has been identified [19] is the “Rashomon effect”, where each counterfactual explanation tells a different (but still true) story to reach a prediction.
Justification-based: In the area of ontological reasoning, there has been a significant amount of work on methods and techniques for computing and working with justifications as a dominant and popular form of explanation [20]. In this context, a justification for an entailment is a minimal subset of the ontology that is sufficient for the entailment to hold. The set of axioms corresponding to the justification is minimal in the sense that if an axiom is removed from the set, the remaining axioms no longer support the entailment. Advantages to this approach include, for example, that they are conceptually simple, they have a clear relationship with the underlying ontology, and there are simple presentation strategies that work well in many settings.

Data-driven: This kind of explanations have recently received much attention because they are very useful in complex AI data-driven models [21, 22], which in general oseek to derive a mathematical function such that the input is a set of values for features or attributes of an entity, and the output is the value of a particular feature of that entity. Typically, this function is obtained by fitting an example entity set (training set), and its main usefulness is the ability to make predictions on the value for the output feature of the entity, provided the process to build the function is carried out successfully. Data-driven explanations are most valuable when the model is complex, in which case the explanation building process is based on deriving another simpler function that approximates the original one.

Statistical. The most likely explanation is not always the best explanation [16], and users benefit most from the identification of causes to explain events, not mere associative relationships; however, these models based on probabilities or statistics are still useful and commonly used in XAI.

Simulation-based. Leverages the implementation of a system to imitate a process of interest in order to analyze the results that arise from running it on one or more input configurations. Some works [23, 24] have focused on exploring the role that these virtual simulations play in social explanations, for example by explaining real-world social phenomena. In general, because simulations play the role of a data-generating experiment, they do not directly provide an explanation; however, they do provide data to evaluate hypotheses within a theoretical framework, which ultimately provides the explanation.

Contextual. Context typically refers to domain information about objects other than the ones explicitly participating as inputs to or outputs generated by the system, such as information about users, situations, and broader environment affecting the computation [25–27]. In [28], the author defines context as a “collection of relevant conditions and surrounding influences that make a situation unique and comprehensible”. Thus, context-aware explanations often include extra information that is not contained in the current described situation but is part of the users’ context.

2.3. HEIST: An Application Framework for XAI in Socio-Technical Systems

As we have mentioned, in this paper we will continue the line of work where we presented the Hybrid Explainable and Interpretable Socio-Technical Systems application Framework (HEIST, for short), adopting it as the central architecture for tackling explainability in ISTSs. HEIST is a framework for guiding the implementation of hybrid XAI-based socio-technical systems via a combination of data-driven and KR-based models. Figure 1-(A) shows an overview of the architecture proposed in [10] (in turn, this model is a generalization of the one first introduced in [29]), which has six components.

- **Data Ingestion**: A variety of data sources feeds this module that is responsible for handling different issues that arise from integration of heterogeneous sources, such as cleaning, schema matching, inconsistency, incompleteness, as well as other higher-level issues such as trust and uncertainty management.
- **Sub-symbolic Services**: There are many tasks that address a variety of problems that are better handled by data-driven services; this module aids in the isolation of application scenarios for each service in order to foster faster deployment, replacement with alternative implementations, and build more reliable explanations.
- **Symbolic Reasoning**: General problems are, in many cases, better handled by high-level reasoning; this module, which can be considered to be the main one in the framework, is fed from both data pre-processed by the Data Ingestion Module and outputs generated by Sub-symbolic Services. In order to take advantage of inference processes, rule-based systems are typically used to carry out high-level tasks such as combining data and low-level knowledge or answering queries using well-defined reasoning mechanisms over highly-structured
knowledge. The reasoning processes implemented here are the main drivers behind answering queries issued by users.

- **Explanations**: As discussed above, different kinds of explanations associated with query answers are implemented in this module, via access to the Symbolic Reasoning (in turn, via the Query Answering module) and Sub-symbolic Services modules.

- **Human in the Loop**: In Social-Technical settings, the system is considered to have failed when it does not adequately consider its users’ demands. This module seeks to mitigate this problem and improve system performance with iteratively administered feedback provided by human users with queries, answers, explanation requests, explanation scoring, and ranking of data sources by usefulness, among other options.

- **Query Answering**: Finally, this module is responsible for answering user queries, which requires coordinating the execution of the rest of the modules.

Having introduced the HEIST framework, in the next section we show how a particular instantiation can be derived to produce a system for a specific use case.

---

3 Human-in-the-loop or HITL is defined as a system/model that requires different degrees of human interaction.
3. Instantiating the HEIST Framework: The NetDER-HS System

In this section, we sketch how the HEIST framework can be instantiated for the implementation of explainable ISTSs. We begin with a brief introduction to a previously-proposed framework called NetDER, which was developed as a tool for implementing ISTSs based on the concept of automated hypothesis generation; HEIST is a generalization of NetDER where the symbolic and sub-symbolic modules are architecturally distinguished. Then, we show how the NetDER-HS system for detecting hate speech in online platforms is derived.

3.1. The NetDER Model

Figure 1 shows how the HEIST architecture generalizes NetDER as proposed in [29]—note that NetDER does not make explicit reference to sub-symbolic services as we do in HEIST. HEIST’s Symbolic Reasoning module is instantiated in NetDER by choosing a particular knowledge representation language. Furthermore, part of the knowledge base is assumed to encode a network of users in which diffusion processes of interest take place. This leads to two sub-modules:

- **Ontological Reasoning**: Stores the knowledge database both for background knowledge (ontological data, rules, and constraints) as well as for network knowledge (nodes, edges, and their attributes). The language chosen for this module is that of existential rules (also commonly referred to as Datalog+/−) [30]. For the purposes of carrying out reasoning tasks, the ontological reasoning sub-module can pose queries to the Network Diffusion sub-module, which offers a supporting role.

- **Network Diffusion**: Responsible for modeling dynamic aspects of the network in the form of diffusion processes and performing condition checks over the network state. There are many ways in which this module can be implemented; here we will assume that a rule-based language is used for such processes, which answer queries over the network required by the Ontological Reasoning module.

Formally, NetDER knowledge bases in the Symbolic Reasoning module are defined over a relational schema \( \mathcal{R} \) of the form \( \mathcal{KB} = (D, G, \Sigma, P) \) such that \( D \) is a set of atoms, \( G \) is a set of facts to allow us to define the initial network knowledge, \( \Sigma \) a set of extended existential rules incorporating network knowledge, and \( P \) is a set of global and local diffusion rules where the former provide a summarized view of the global network state, while the latter are used to guide the evolution of the network state that will be necessary in the forecasts. The main reasoning task is carried out by a query answering mechanism; NetDER queries are of the form:

\[
Q(X) = \exists \rho(X_1, Y_1) \land \phi(X_2, Y_2) \land \gamma(X_3, Y_3) : [t_1, t_2]
\]

where \( \rho(X_1, Y_1) \) is a conjunction associated with ontological knowledge, \( \phi(X_2, Y_2) \) is a conjunction associated with local network knowledge, \( \gamma(X_3, Y_3) \) is a conjunction related to global network knowledge, and \( [t_1, t_2] \) is a time interval where these conditions should hold. Finally, each entity in the relational schema may have associated one or more sub-symbolic services, which may be invoked in order to estimate the value of a property of interest of that entity.

This combination of data-driven and KR-based models is very powerful for achieving explainability, since it allows to understand the logic behind the outputs that the query answering component can provide. There are many ways in which explanations can be derived; here, we consider two main approaches, which can also be combined. First, as mentioned above, having access to the ontological knowledge is often a useful starting point in understanding the system’s outputs. Second, it may be necessary to access dynamic aspects of the network diffusion process, such as the network state. In Section 4, we go into further detail by proposing several types of explanations that can be generated based on these components, particularly taking advantage of combining both knowledge- and data-driven explanation techniques.

3.2. A System for Detecting Hate Speech in Social Platforms

In this section, we instantiate HEIST/NetDER to derive a system for detecting hate speech in social platforms—an overview of this instantiation is shown in Figure 1-(B), and we call the resulting system NetDER-HS. We
describe a simple use case where we have a typical social platform with a set of users that share information by posting and reposting content, or replying to posts from other users. In particular, we define a set of rules in the Ontological Reasoning module to work in a hate speech detection setting. Towards this end, we need to provide the details of the main components; as a first step, we assume that the Data Ingestion module feeds data into the ABox (ontological database) as well as the encoding of the network based on data about:

- **Posts**: Any content posted on social media platforms.
- **User profiles**: Information about users in a social platform (such as city, age, interests, etc.).
- **User relations**: Information about the existing connections between different user profiles.

We assume that the Data Ingestion module feeds the database available in the Symbolic Reasoning module with data in the following schema:

- **node(U)**: U is a node of the graph, which in this case represents a user of the social platform.
- **edge(U1, U2)**: Nodes U1 and U2 are connected, representing relations such as follow, friendship, fan, etc.
- **post(P)**: content P has been posted.
- **posted(U, P)**: User U posted P.
- **hs_level(P, L)**: Post P is considered to contain hate speech by some sub-symbolic service with certainty level $0 \leq L \leq 1$.
- **early_poster(UID, P)**: UID is among the first users to share post P.
- **hyp_hatespeech(P)**: Hypothesis that post P contains hate speech.
- **hyp_is_resp(UID, P)**: Hypothesis that user UID is responsible (perhaps unwittingly) for sharing post P.
- **hyp_hater(UID)**: Hypothesis that user UID is a “hater”—we use this name to refer to someone who knowingly posts hateful comments.
- **repeatedly_banned(UID)**: User UID has been banned from the platform multiple times.
- **avg_banned(T)**: T is the average number of bans per user on the platform.
- **popular**: A node is considered to be a popular user (for instance, because their posts receive a lot of attention, and they have many followers).
- **repost(P)**: Post P is a reposted message.
- **offensive_language(P, L)**: Post P is believed to contain offensive language with certainty level $0 \leq L \leq 1$ by some sub-symbolic service.
- **targeted(P)**: Post P is targeted to an individual or group of people.
- **hs_context_level(P, L)**: Post P is considered to contain hate speech with certainty level $0 \leq L \leq 1$ taking contextual information into account.
- **hs_level(P, L)**: Post P is considered to contain hate speech with certainty level $0 \leq L \leq 1$.

Note that **hs_level/2, hs_context_level/2, and offensive_language/2** are atoms derived from external data-driven classifiers (such as the tools developed by [31] and [32]). Hence, these can be seen as wrappers for a call to a sub-symbolic service. This easily allows to tune different thresholds required in the rules to be instantiated, replace one classifier with another, combine different classifiers (ensembles), and use machine learning-based predictors to solve specific tasks.

The Ontological Reasoning sub-module contains the logical rules shown in Figure 2; we provide intuitive explanations for each rule next. Roughly speaking, rule $r_1$ states that given a post that the classifier detects as offensive with confidence level $L$ (greater than the threshold 0.5) and is targeted to a person or group of people, a hypothesis is created that $P$ is hate speech with high danger level (red); on the other hand, $r_2$ expresses that a hypothesis that $P$ is hate speech with medium danger level (orange) is created if $P$ is an offensive post (that is, the targeting condition is removed) with a lower threshold for its confidence level. Rule $r_3$ generates a hypothesis that $P$ is hate speech with danger level red if $P$ is detected as hate speech with confidence $L > 0.5$; rule $r_4$ is a variant that uses a context-based classifier when $L > 0.2$ and $L < 0.5$. If the user who posted $P$ is suspected to be a hater, then the rule $r_5$ generates hypotheses for the existence of hate speech with lower danger level (yellow). Observe that the sixth and eighth rules produce hypotheses stating the existence of haters; being the user UID responsible for disseminating two hate speech posts is enough for generating the hypothesis that UID is a hater, whereas having been repeatedly banned is
Ontological Rules:

\( r_1 : post(P) \land offensive_language(P, L) \land (L > 0.5) \land targeted(P) \rightarrow hyp_hatespeech(P, red) \)

\( r_2 : post(P) \land offensive_language(P, L) \land (L > 0.2) \rightarrow hyp_hatespeech(P, orange) \)

\( r_3 : post(P) \land hs_level(P, L) \land (L > 0.5) \rightarrow hyp_hatespeech(P, red) \)

\( r_4 : post(P) \land hs_level(P, L) \land (L_1 > 0.2) \land (L_1 < 0.5) \land hs_context_level(P, L_2) \land (L_2 > 0.5) \rightarrow hyp_hatespeech(P, orange) \)

\( r_5 : post(P) \land reposted(UID, P) \land hyp_hater(UID) \rightarrow hyp_hatespeech(P, yellow) \)

\( r_6 : hyp_is_resp(UID, P_1) \land hyp_is_resp(UID, P_2) \land (P_1 \neq P_2) \rightarrow hyp_hater(UID) \)

\( r_7 : hyp_hatespeech(P, L) \land early_poster(UID, P) \rightarrow hyp_is_resp(UID, P) \land \langle viral(P), [0.7, 1] \rangle \)

\( r_8 : hyp_is_resp(UID, P_1) \land repeatedly_banned(UID) \rightarrow hyp_hater(UID) \)

\( r_9 : hyp_is_resp(UID, P_1) \land banned(UID, T_2) \land avg_banned(T_2) \land (T_1 > T_2) \rightarrow \) repeatedly_banned(UID)

Diffusion Rules:

\( loc_rule_1 : repost(P, \langle popular, [0.0, 0.5] \rangle) \rightarrow (T, \langle repost(P), [1, 1] \rangle \land \langle popular, [1, 1] \rangle, if) \)

\( glob_rule_2 : viral(P) \leftarrow repost(P), avg \)

Fig. 2. Rules in the Ontological Reasoning Module.

also a reason for generating the hypothesis that UID is a hater. Finally, the seventh and ninth rules can be read as:

“If post P is suspected to contain hate speech (with any danger level), and user UID is considered to be an early poster of P that is flagged as viral on the network (with confidence at least 0.7), then there exists a hypothesis that UID is responsible for disseminating hate speech”, and “the hypothesis that UID is repeatedly banned is created if the number of bans for UID’s account is above the average number of bans per user”.

For the Network Diffusion sub-module, we could implement a set of logical diffusion rules with the purpose of guiding the evolution of (uncertain) knowledge about the network—an example of a language for expressing such rules is MANCaLog [33]. For this particular example, Figure 2 shows two diffusion rules, loc_rule_1 and glob_rule_1, where the former is a local diffusion rule that intuitively states “users that are not popular with certainty and have popular neighbors who reposted P with certainty, will update the belief that P should be reposted, according to influence function if". In this case, we assume that function if always returns the interval [1, 1], meaning that P is likely to be reposted by the node in the next time point. Likewise, glob_rule_1 is a global diffusion rule that allows us to flag a post as viral based on the forecasting of reposts for each node. Note that this rule involves a function avg that will be used to forecast the viral state of each post and it returns the required value by computing an average for the values associated with each of the node’s reposts. We refer the reader to [29] and [34] for further discussions about diffusion rules.

Note that both the ontological knowledge base and network database can be beneficial for the construction of complex explanations for the outputs generated by NETDER-based systems. In the next section, we will study several explanation types, with a special focus on explainability in NETDER—the NETDER-HS system will be adopted as the use case throughout the rest of the paper in order to show concrete examples.

4. Towards Concrete Explanations in the Context of HEIST

We now focus on producing concrete types of explanations in the context of systems implemented using the HEIST application framework; in order to be able to have a detailed discussion, we assume the pre-instantiation of HEIST into NETDER as discussed in the previous section. We begin by proposing a process by which a user can choose and combine explanation styles as required; this is done by selecting among five different sources
STEP 1
Please choose at least one of the following sources:

<table>
<thead>
<tr>
<th>ONTOLOGICAL</th>
<th>DIFFUSIONAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available options:</td>
<td>Available options:</td>
</tr>
<tr>
<td>• Assertional-based (A)</td>
<td>• Process-focused (P)</td>
</tr>
<tr>
<td>• Terminological-based (T)</td>
<td>• Result-focused (R)</td>
</tr>
<tr>
<td>• Full Ontological (FO)</td>
<td></td>
</tr>
</tbody>
</table>

STEP 2
Please choose at most two of the following styles:

- [ ] Contrastive
- [ ] Counterfactual
- [ ] Justification-based
- [ ] Data-driven
- [ ] Statistical
- [ ] Simulation-based
- [ ] Contextual

Fig. 3. Wireframe of the interface for the proposed explanation customization process.

of explanatory knowledge (introduced in Section 4.1), and then requesting the specific style(s) (as discussed in Section 4.2).

Build Your Own Explanation

Figure 3 illustrates—in a wireframe layout—a general graphical representation of a process for choosing custom-built explanations. This process can be outlined as a two-step sequence:

Step 1: Select the information sources to be used in building the explanation. This choice may include a single source or all of them—this is a first filter over the explanation styles that will be available in Step 2.

Step 2: Select the style(s) required for delivering the explanation. As not all combinations are compatible, the system can simply gray out the ones that cannot be chosen as the user makes selections.

So, as an example, we might be interested in a counterfactual explanation that is built based solely on terminological knowledge. In the next section, we go into greater depth about the different sources and explanatory styles shown in Figure 3, developing the single-style options into more detail. Later, in Section 4.3, we will analyze the combination of pairs of explanatory styles.

4.1. Sources of Explanatory Knowledge

We now study the range of possibilities that we have for leveraging the different components of an ISTS based on the HEIST application framework. We consider four possible knowledge sources and their associated explanations: Terminological-based (T), Assertional-based (A), Process-focused (P), and Result-focused (R); a fifth option, which combines the first two, is the Full Ontological (FO) source. Next, we describe them in more detail.

Terminological-based (T) explanations. In our context, explanations may seek to benefit by accurately extracting information and insights contained in the ontological rules/axioms; leveraging this source allows us to understand why the system provides an answer for a specific query by highlighting the rules involved in its derivation. For example, in a social platform setting like the one in the previous section, pointing to the specific rules used in
### Primitive Explanation Styles

<table>
<thead>
<tr>
<th>Sources</th>
<th>A</th>
<th>T</th>
<th>FO</th>
<th>P</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Contrastive</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2. Counterfactual</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3. Justification-based</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>4. Data-driven</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>5. Statistical</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>6. Simulation-based</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>7. Contextual</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Fig. 4. Sources available to each primitive explanation style. Checkmarks indicate availability, crosses indicate non-availability, and question marks are used when sources are potentially available, but exploring the specific conditions is outside the scope of this work.

reaching a conclusion that a user is suspected to be a “hater” may be a useful way to convey to the system’s user why this inference was made.

**Assertional-based (A) explanations.** Atomic formulas (or assertional axioms) are basic objects that in our context refer to elements present in both the ontological database and network database. Choosing this source thus allows to incorporate elements from this basic set which, as we will see, is often connected to several explanation styles arising in data-driven AI techniques. For example, an explanation may use the data available on a social platform user to show why they are classified as potentially malicious.

**Process-focused (P) explanations.** In this case, explanations are designed with the aim of shedding light on how the network model evolves as the execution of the diffusion process progresses from the initial state to the final state (when the model is ready to make a prediction). For example, if a user posts a comment containing offensive language, a rule may be fired incrementing the probability that the user is a hater (as in the use case above), producing an evolution of the network model modifying the initial state. Then, having a user with this probability above a certain threshold may fire another rule issuing a prediction that the user will likely post such content again in the future, and should be flagged with a first warning, in this simple case already reaching the final state of the network model. Explanations of this kind would then highlight the rules used to update the network model during the process; in general, if the diffusion process is not rule-based, explanations can highlight the transitions that the model goes through.

**Result-focused (R) explanations.** These explanations focus on the final result of the diffusion process over the network, which can be seen as a forecast of the network’s evolution over time. For example, an explanation as to why a post has been flagged may highlight how the post will “go viral” by showing the number of users it is expected to reach in the next three days.

### 4.2. Single-style (Primitive-only) Explanations

In this section, we present additional details regarding the primitive explanation styles discussed in Section 2.2. We provide initial details on each primitive style, providing examples in each case depending on the sources that are available to each, as presented in Figure 4.

**Contrastive explanation.** Consider the following elements involved in this explanation style:

- A NETDER knowledge base $KB = (D, G, \Sigma, P)$.
- A NETDER query $CQ(X)$ where $X = X_1, \ldots, X_n$.
- Two Boolean NETDER queries $BCQ$ and $BCQ'$ of the form $BCQ = CQ(X_1 = v_1, \ldots, X_i = v_i, \ldots, X_n = v_n)$ and $BCQ' = CQ(X_1 = v'_1, \ldots, X_i = v'_i, \ldots, X_n = v'_n)$, where it holds that $v'_i \neq v_i$ for some $1 \leq i \leq n$ and $\text{similar}(BCQ, BCQ')$.  

A contrastive explanation for an answer to BCQ over KB is the pair \((BCQ, a), (BCQ', a')\), where \(a, a' \in \{Yes, No\}\) and the answers are to BCQ and BCQ', respectively.

Note that function \(similar(Q_1, Q_2)\) requires implementation in order to yield a similarity value given two queries (cf. Class ContrastiveStyle in Section 5).

**Example:** Consider the application domain introduced in Section 3.1, and the user query \(hyp\_hater(john)\), which has answer Yes. Assuming that \(similar(hyp\_hater(john), hyp\_hater(paul))\) holds, the contrastive explanation is: \((\langle hyp\_hater(john), Yes\rangle, \langle hyp\_hater(paul), Yes\rangle)\), which can intuitively be interpreted as “user John is a hater because user Paul is similar to John, and he is a hater, too”.

**Counterfactual explanation.** This style yields explanations built using knowledge that is not included in any knowledge source. More concretely, consider a Boolean query \(Q\) and a NETDER knowledge base \(KB = (D, G, \Sigma, P)\). A counterfactual explanation for an answer to \(Q\) over \(KB\) is based on both available and not available knowledge—that is, it produces \(KB' = (D', G', \Sigma', P')\) by changing elements in \(KB\) (such as data and/or rules) and that is also used to answer query \(Q\).

**Example:** Continuing with the example illustrated in Figure 2, let \(hyp\_hatespeech(post_1, yellow)\) be a query, which we assume has answer “No”. A counterfactual explanation to this answer can be generated by adding new information to the knowledge base, such as the fact that a user who posted \(post_1\) is identified as a hater. Furthermore, suppose that with this new information rule \(r_1\) can now be applied and the answer for our query now becomes “Yes”.

A counterfactual explanation in this case could then be “there would be a hypothesis for hyp\_hatespeech(post_1, yellow) if it were the case that the user who posted post_1 is suspected to be a hater”.

**Justification-based explanations.** In our setting, we assume we have a Boolean NETDER query \(Q\), a NETDER knowledge base \(KB = (D, G, \Sigma, P)\), and if the answer for \(Q\) over \(KB\) is “Yes”, then a justification-based explanation for this answer is another NETDER knowledge base \(KB' = (D', G', \Sigma', P')\) such that:

- \(D' \subseteq D\),
- \(G' \subseteq G, \Sigma' \subseteq \Sigma\),
- \(P' \subseteq P\),
- the answer for \(Q\) over \(KB'\) is also “Yes”, and
- for all \(KB'' = (D'', G'', \Sigma'', P'')\) if \(D'' \subseteq D', G'' \subseteq G', \Sigma'' \subseteq \Sigma',\) and \(P'' \subseteq P'\), then the answer for \(Q\) over \(KB''\) is “No”—that is, it must include only the minimal necessary knowledge.

Note that justification-based explanations only make sense when the answer is ’Yes’, since otherwise this means that there does not exist a minimal subset of knowledge to support the query. Figure 4 indicates that this kind of explanation is available with sources from the ontology (A, T, and FO); in the case of P and R-sourced explanations, the same criteria could be applied, though this will depend on the specific language used to implement the diffusion process. Exploring this issue further is the topic of future work.

**Example:** Consider the ontological rules presented in Figure 2, and let \(hyp\_hatespeech(post_1, red)\) be a boolean query. Suppose we can create a hypothesis for \(hyp\_hatespeech(post_1, red)\), which can be obtained by applying rule \(r_1\). From our definition of justification-based explanation, we can generate an explanation with \(r_1\) and the set of atoms \(\{post_1, offensive\_language(post_1, 0.6), targeted(post_1)\}\), which we assume holds. From the point of view of an ontology engineer, this type of explanation would for instance help carry out debugging tasks by pinpointing what causes \(hyp\_hatespeech(post_1, red)\) to hold.

**Data-driven explanations.** In this context, we have the following central elements:

- A NETDER knowledge base \(KB = (D, G, \Sigma, P)\) over the relational schema \(\mathcal{R}\).
- A set of entities \(E \subseteq \mathcal{R}\).
- An instance \(e(v_1, \ldots, v_n, v_{n+1})\) of an entity \(e \in E\).

\(^{4}\)Though it is not strictly necessary for the answers over \(KB\) and \(KB'\) to be different, it is often useful for this to be the case.
– A set of sub-symbolic services $S$, and a sub-symbolic service $s_1 \in S$ associated with the entity $e$.
– A set $E$ of instances $e(v'_1, \ldots, v'_n, v'_n^{+1})$ of the entity $e$.
– A threshold $\theta_1 \in [0, 1]$, and a comparison operator $op \in \{ >, <, =, \neq, \geq, \leq \}$.
– A value $v_{n+1} \in [0, 1]$ that results from invoking $s_1$ over $(v'_1, \ldots, v_n)$.

Then, we say that a data-driven explanation of the answer for the query $e(v_1, \ldots, v_n, v_{n+1}) \wedge (v_{n+1} \ op \ \theta_1)$ over $KB$ is an explanation based on a function $f$ such that for all $e(v'_1, \ldots, v'_n, v'_n^{+1}) \in E$ we have $f(v'_1, \ldots, v'_n) = v'_n^{+1}$ and $v'_n^{+1} \in [0, 1]$ is also the result of invoking $s_1$ over $(v'_1, \ldots, v'_n)$.

Note that, as shown in Figure 4, this primitive style is only available for the assertional source.

**Statistical explanation.** This kind of explanation is likely the most general and flexible of the primitive styles we consider, since it allows us to summarize in a numeric way the reasons involved in obtaining a specific output of $Q$ over $KB$. Hence, a data-driven explanation could rely on a function $f$ obtained from variants of the entity offensive_language and the result of predictions for these variants made by $s_1$; in this case, it can be used in the provided explanation in order to unveil the most important words involved in the assertion “offensive_language is used in post”.

**Example:** We can assume that a data-driven explanation request is posed in order to know the reasons behind the answer “Yes” for the query offensive_language(post, level) $\wedge$ (level $>$ 0.5), where post is a value based on textual content using offensive language with confidence value level predicted by the sub-symbolic service $s_1$. Hence, a data-driven explanation could rely on a function $f$ obtained from variants of the entity offensive_language and the result of predictions for these variants made by $s_1$; in this case, it can be used in the provided explanation in order to unveil the most important words involved in the assertion “offensive_language is used in post”.

**Contextual explanation.** Consider a Boolean $\text{DER}$ knowledge base $KB = (D, G, \Sigma, P)$; a contextual explanation of an answer to $Q$ over $KB$ is an explanation based on numeric indicators from functions that summarize the occurrence of data related to the answers of $Q$ over $KB$. Examples of such functions required for these explanations are the well-known aggregation functions such as max, min, classical statistical functions such as mean, median, standard deviation, or even more complex ones.

**Example:** Consider again the query hyp_hater(paul), and assume that the answer to this query is “Yes”. Examples of these explanations can include outputs of functions used to calculate the number of occurrence of posts, bans, comments, and specific connections between users such as follow, friendship, likes, etc. More concretely, network atoms associated with the current network state—encoded by atoms as node, edge, posted among others—offer interesting alternatives when computing such statistics.

**Simulation-based explanation.** Here, we have a Boolean $\text{NETDER}$ query $Q$ and a $\text{NETDER}$ knowledge base $KB = (D, G, \Sigma, P)$; an simulation-based explanation of an answer to $Q$ over $KB$ is built based on imitating the evolution over time of a network diffusion process that provides insight as to why it is possible to answer $Q$ using $KB$. As indicated in Figure 4, this primitive style is only available in the process- and result-focused sources.

**Example:** An explanation based on a forecasting model about the evolution of content in the network may explain why a post has been flagged as viral. Consider the diffusion rule $\text{diff_rule}_1$ presented in Figure 2; a possible explanation may be to show how a diffusion process models the future spread of hate content after three days, indicating estimates of the percentage of users that will repost such content.

In requesting an explanation, our proposal provides facilities for generating explanations that could combine several styles and knowledge sources. In order to better understand these combinations, we carry out here a preliminary
study about which explanations can be built using the styles and sources addressed in this paper. Our main findings are summarized in Figure 5—note that we do not consider all possibilities of combinations between available sources and explanation styles; in these first steps towards understanding combined explanations, we analyze how to combine two single-style explanations at a time.

Different approaches could be developed for combining explanation styles—this is in general not a simple endeavor due to several characteristics, such as contemplating the combined explanation’s goal, stakeholders, application domain, among others. For the purposes of this work, we identify three basic options:

**Policy 1:** Obtain each of the selected styles independently and then deliver them together.

**Policy 2:** Generate a systematic composition of explanations by sequentially applying styles, starting from the initial source(s) of explanatory information and using the output of one as the input to the next.

**Policy 3:** Implement an ad hoc combination when explanations cannot be addressed independently.

Policy 1 can be simply applied by taking the information from Figure 4 and checking if all the chosen styles are compatible with the selected sources—this is the policy we include in our initial implementation (cf. Section 5). We now provide a preliminary discussion of the feasibility of applying Policy 2 with two styles; Policy 3 will be addressed in future work.
Combining Two Different Styles

In order to apply Policy 2, we need to understand what combinations are useful and comprehensible to make sense of the resulting explanation. In the following, we discuss the central aspects that emerge from considering the pairwise combinations shown in Figure 5; in some cases, the scope of the discussion may be limited by the fact that concrete implementation details of each particular style, as well as factors such as whether or not the explanation generation process is domain-specific. Our discussion will be organized according to the blocks in which the table in Figure 5 is separated.

We begin with an analysis of three groups of combinations that share an important feature that causes order to be important; in particular, in each case there is a primitive style that generates elements to be used in the explanation that can later naturally be used as inputs to other styles:

- **Counterfactual + another primitive style (Styles 8–13):** The counterfactual style involves deriving another knowledge base, which can then be taken as input for the other style.

- **Contrastive + another primitive style (Styles 14–18):** The contrastive style provides a pair of queries and their answers, which can in turn be used by the second style.

- **Justification-based + another primitive style (Styles 19–22):** The justification-based style identifies a subset of the knowledge base, which—analogously to the combination with counterfactuals—can be taken as input by the other style.

The table in Figure 5 shows these resulting natural orders to derive combined explanations. Consider the following example for explanation style 14 (Contrastive + Justification-based) that arises from combining the examples included in Section 4.2. Upon requesting the contrastive style (cf. Page 12), the system yields

\[
\langle (\text{hyp}_\text{hater}(\text{john}), \text{Yes}), (\text{hyp}_\text{hater}(\text{paul}), \text{Yes}) \rangle
\]

If a justification-based explanation is then requested over this output, one possible explanation would be that \(\text{hyp}_\text{hater}(\text{john})\) was derived using rules \(r_7\) and \(r_8\), while \(\text{hyp}_\text{hater}(\text{paul})\) was derived via \(r_6\) and \(r_8\) (and the atoms that caused each rule to fire).

- **Styles 23–27:** In this group, we have four possibilities. In the first two, we have Data-driven with either Statistical or Contextual styles; for the former, the most natural order is to first apply the Data-driven style, which can then be fed to the Statistical style, while for the latter, both orders could in principle be applied: Data-driven \(<\) Contextual, or the other way around. Similarly, for Style 26, the Statistical style is most naturally applied last, while in the case of Style 27 both orders are possible. For an illustrative example for style 23 (Data-driven + Statistical), consider the example from Section 4.2 (cf. Page 13) where the user requests a data-driven explanation for \(\text{offensive_language}(\text{post}, \text{level}) \land (\text{level} > 0.5)\) and is shown the set of most relevant words involved in that assessment. If an additional Statistical explanation is requested, the system may provide the information regarding the percentage of times in which posts containing those words were correctly classified as offensive.

Finally, in Style 28 the most sensible order is Simulation-based \(<\) Contextual, given that a simulation may be enhanced by contextual information.

5. The Main Implementation Tools

In this section, we discuss in detail the software engineering aspects involved in a preliminary implementation of our general framework, and how it can be used in practice. The content is organized as follows:

- Detailed presentation of the development of the HEIST application framework, providing publicly-available code specifying its structure.

- Specification of how HEIST can be extended to obtain the NetDER model.

- Illustration of the workflow involved in the explanation request activity, which is the main focus of this work.

The source code for this initial version of the HEIST application framework is publicly available at: https://github.com/jnparedes/HEIST.
5.1. The HEIST Framework

Figure 6 shows a high-level view of the class diagram for the HEIST framework (we include the detailed version of this diagram in Appendix A); this diagram corresponds to a more detailed view of Figure 1 (A). In the following, we describe the most important links between the classes, which appear as relations in the diagram. For the purposes

Fig. 6. High-level view of the class structure of the HEIST application framework (for detailed view, see Figure 9 in Appendix A).
of this work, we assume that the symbolic reasoning module will be implemented as a rule-based system (i.e., the knowledge base is comprised of a set of rules and a set of atomic formulas referred to as a database).

One of the main components in HEIST is the Data Ingestion module, which is in charge of creating the knowledge base and also communicates with the Sub-symbolic Services module in order to both train new data-driven tools and make use of others already in place. The Symbolic Reasoning module implements a series of reasoning tasks, each of which involves a specific inference mechanism that functions by leveraging specific knowledge sources. The Query Answering module is in charge of delivering the main system functionality, which depends primarily on the Symbolic Reasoning module (which, in turn, is supported by the Sub-symbolic Services). The main client of the Query Answering module is the User, who issues queries and may also request explanations for their answers. In that case, the Explanations module interacts with: (i) the Query Answering module, in order to obtain the available sources of knowledge; (ii) the Sub-symbolic Services module, to have access to the sub-modules that are related to the query answers; and (iii) the User, in order to allow them to choose the source(s) and style(s) for the requested explanation.

Finally, in designing this class structure, the Strategy design pattern was applied for the ReasoningTask, Explanation, Source, and Style classes. For the StyleVisitorFactory class, a combination of the AbstractFactory and Visitor patterns were applied.

5.2. The NetDER Model as an Extension of the HEIST Framework

Figure 7 shows the high-level class diagram for the instantiation of HEIST that produces the NetDER model (as presented in [29])—this diagram corresponds to a more detailed view of Figure 1 (B). Compared to the previous section’s diagram, here we can see that this model includes the result of making several implementation choices:

– Reasoning Tasks: Chase (for query answering with existential rules) and Diffusion (which is used to represent and query diffusion processes in networked data).
– Knowledge Sources: The Source class is used to implement the knowledge sources, of which we have four: Terminological and Assertional, to be used by Chase, and Process-focused and Result-focused, to be used by Diffusion.
– Explanation Styles and their associated Explanations: we have one class for each primitive style discussed in Section 4.2 and corresponding explanation:

* ContrastiveStyle and Contrastive
* CounterfactualStyle and Counterfactual
* Justification-basedStyle and Justification-based
* Data-drivenStyle and Data-driven
* Data-drivenStyle and Data-driven
* StatisticalStyle and Statistical
* Simulation-basedStyle and Simulation-based
* ContextualStyle and Contextual

In this preliminary version, we only consider the implementation issues for combining single-style explanations via Policy 1 (cf. Section 4.3); more complex combined-style explanations will be incorporated in a later version.
Fig. 7. High-level view of the class structure of the NuTDER instantiation of HEIST.
5.3. The Explanation Request Workflow

Figure 8 shows interaction diagrams for the main activity of interest for this paper: the explanation request issued by a user. The diagram at the top shows the main steps: a user carries out an explanation request for a query answer, which implies that a User object receives a message that invokes the request(ans,q) method, and then the sources are chosen in order to provide the necessary ingredients to build the explanations. Then, the explanation styles are chosen, which allow us to define the particular nature of the required explanation.

The diagram at the bottom of Figure 8 shows a more detailed view of what goes on in the explain(ans, q) method in ExplanationsModule: after sources and styles are chosen, an ExplanationsModule object receives an invocation of its explain(ans,q) method associated with a query and an answer. The StyleVisitorFactory classes relevant to the user’s choices are then in charge of creating the necessary style objects; the chosen knowledge sources also affect the process by linking the relevant source objects—this essentially implements the source-style associations illustrated in Figure 4. After these steps, the ExplanationsModule object will request from each style object the associated explanations, which are returned to the user as a collection of Explanation objects.
6. Related work

In considering the existing literature, we will begin by discussing the related work in the area of explainability, particularly in data-driven and knowledge-based systems. Explainability is a vast, multidisciplinary area that has especially received much attention in the last five years or so, and providing a complete survey is outside of the scope of this work. After covering XAI from the two main sides that are involved in our model (knowledge-based and data-driven), we briefly discuss XAI in cybersecurity, which is the example application domain used throughout this work.

6.1. Knowledge-based Explanations

Knowledge-based systems were conceived as tools to support human decision-making, and generally consist of a knowledge base encoding the domain knowledge, usually modeled as a set of rules, a database, and a reasoner that makes use of these components to answer user queries. Though traditionally logic-based reasoning formalisms have been touted as more closely related to the way humans reason, and therefore more inherently explainable, different kinds of explanations are essential to the interaction between users and knowledge-based systems, as discussed in this paper. Thus, questions like what a system does, how it works, and why its actions are appropriate may not be immediately answerable even for logic-based approaches.

Many works, such as [20, 35, 36], focus on explaining a system’s decisions from a range of several different explanations types, providing various levels of assistance. In this regard, when requiring domain information to explain a decision, structured forms of domain information like ontologies are in many cases particularly well suited. For example, there has been an important body of research directed towards the area of explanation and ontology debugging related to the Web Ontology Language (OWL); in particular, work has focused on a specific type of explanation called justifications [20, 37, 38]. Approaches like [39] have focused on implementation aspects by presenting libraries and computational tools for working with justification-based explanations of entailments in OWL ontologies. The main focus of research in this area has been on explanation for the sake of debugging problematic entailments, such as class unsatisfiability or ontology inconsistency. Recent successes in machine learning technologies [40] have brought more attention to explainability, especially on how to integrate and use domain knowledge to drive the explanation delivery process and how this influences the understanding of explanations by users. Our proposal is part of the same general effort to enhance logic-based formalisms with knowledge obtained from the use of machine learning tools. In this regard, examples of advantages of using HEIST include adaptability and interoperability with data models implemented as a service in the sub-symbolic module. Other specific works on knowledge-based explainability include approaches based on argumentation [41], knowledge graphs [42], and answer set programming [43].

6.2. Data-driven and Hybrid Explanations

In the last decade or so, with the availability of vast volumes of data, machine learning algorithms have seen a boom in popularity—however, these models are typically seen as hard to understand, and there has thus been a significant focus on making them explainable. While some models are considered explainable by design (such as decision trees), others do not reveal sufficient details about how their outputs are generated, resulting in an opaque (also referred to as black box) decision model. This has led to the development of a variety of approaches for deriving explanations of opaque models, for both fully autonomous and human-in-the-loop systems. The literature on data-driven XAI is vast, and although there is no clear general consensus on what constitutes a good explanation, many taxonomies commonly used to classify explainability methods cover the following dimensions:

- **Method:** There are two main types of methods to provide explanations. Local ones give explanations via individual instances or groups of nearby instances, while global ones describe the behavior of models as a whole.
- **Stakeholder:** There are three large groups of stakeholders: Developers and AI Researchers who create AI systems, Domain Experts who are specialists in the area of application, and Lay Users who come into contact with the system’s functionalities.
These and other aspects are discussed in greater depth in surveys such as [21, 22].

Recently, Artificial Intelligence efforts have evolved to hybrid systems that employ both data-driven and logical reasoning techniques. With increasing adoption of these systems, there is a need for approaches that tackle explainability as a primary consideration, fitting end user needs to specific explanation types and the system’s AI capabilities. Following this idea, [44] presents an Explanation Ontology that treats explanations as a primary component of AI system design, providing a semantic representation that can help system designers to capture and structure the various components necessary to enable the generation of user-centric explanations in their systems. The ontology captures several classes and properties from [45], where the authors proposed an Explanation Ontology Design Pattern to represent explanations. Another interesting work is that of [46], which shows an extension of the Explanation Ontology to model explanations in the food domain.

Finally, there are several similarities and differences between our work and the approaches mentioned above. In this setting, our approach can be valuable for building systems with hybrid reasoning mechanisms capable of generating different explanation types, including combined types like the ones explored, and allowing human-in-the-loop interactions. We provide the basic building blocks for the implementation of explainable intelligent socio-technical systems based on NETDER, a particular case of the HEIST application framework, which are documented here via class and interaction diagrams and have the corresponding publicly-available source code. Despite their direct application to real-world social platforms, mechanisms to deal with network knowledge-based explanations have not been extensively studied in the literature.

6.3. XAI in Cybersecurity

Explanations are essential to decision-making support, and more importantly when these systems handle critical information (such as in cybersecurity, health, or financial systems, among others). However, XAI in cybersecurity has not received much attention in the literature; we now briefly review differences and similarities between the most salient works related to our approach.

The closest work in spirit to our approach is the proposal presented in [44], which also aims at combining hybrid explanation types by using different forms of knowledge. However, there are several differences with our proposal. First, our model does not focus on introducing a user-centered general-purpose explanation approach, but rather on: (i) analyzing several explanation types for NETDER, which is a system especially designed for the development of systems capable for detecting malicious behavior in social platforms; and (ii) implementing HEIST (proposed up to the design stage in [10]), a general application framework that helps system designers build explainable systems in any domain.

In the area of cybersecurity, a number of relevant research works recently published have focused on explainability. For example, several topics and future challenges to be explored were discussed by Viganò and Magazzeni [6]; their focus is on stressing questions that should be taken into consideration, such as who gives/receives the explanations, what constitutes a good explanation, when the explanation should be given, how to give explanations, where the explanations should be made available, and why the explanations are needed (as discussed in Section 2). In [9], Holder et al. present a use case for capturing these issues, seeking a guide for the development of future explainable systems that users can leverage. Other works are part of a recent surge of research on enhancing the explainability of machine learning techniques; in our domain of interest in particular, Mahdavifar and Ghorbani [7] propose to do this via a rule-extraction from a trained DNN, which is then used to add explanation capabilities for the detection of cyber threats. In [8], Szczepanski et al. develop and evaluate an Explainable Intrusion Detection System based on a combination of ML tools via a microaggregation heuristic. Finally, a particularly closely related work is the one reported by Kuppa et al. in [47], which centers in understanding the security robustness of explainable methods by introducing a taxonomy for XAI methods, capturing various security properties and threat models relevant to the cybersecurity domain. Motivated by this, the authors then present a black-box attack to study the consistency, correctness, and confidence security properties of gradient-based XAI methods.
This line of research takes initial steps towards the implementation of explainable software tools based on the HEIST application framework in general, and the NETDER model in particular, providing decision support via the leveraging of both ontological knowledge and network diffusion processes. To the best of our knowledge, there have not been other works of this kind to date.

7. Conclusions and Future Work

In this paper, we have continued a research line that began in [29] with the proposal of the NETDER architecture as a general model for reasoning about malicious behavior in social platforms and was recently generalized to the HEIST application framework for explainable query answering in cybersecurity domains, presenting its design in [10, 11]. Here, we have implemented this framework and focused on further developing the explainable query answering functionality by: (i) identifying sources of explanatory knowledge (two ontology-based and two complementary ones); (ii) identifying seven primitive explanation styles from the XAI literature plus 21 combined styles, and studying how they can be adapted to work with the source of knowledge; (iii) developing a “build-your-own explanation” approach to allow users to choose the source(s) and style(s) they want to base their explanation requests on for a specific query. We developed a proof-of-concept use case based on hate speech detection in social platforms, which we used as a running example to illustrate our discussions. Finally, we developed and made publicly available a preliminary implementation of our application framework and documented its use for the explanation request functionality.

There remains much work to be done as part of our efforts to materialize the vision proposed in [3] of moving “from ontology-driven information systems to ontology-driven socio-technical systems”. The goal of this paper was to take a systems engineering perspective on incorporating explainability functionalities—concerns regarding the provision of adequate and high-quality explanations are outside the scope of this work since they need to be addressed for specific combinations of concrete systems and domains in order to avoid issues like the ones discussed in [48]. Ongoing and future work involve studying the combination of explanation styles in greater depth in order to improve the coverage of the vast space of potential user requirements regarding explainability, continue development of our application framework to incorporate combined explanation styles, and testing its deployment in real-world applications.

References


Appendix A. Additional material

Fig. 9. Detailed class diagram (low-level view of Figure 6).