IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 29, 2018, accepted May 22, 2018, date of publication June 5, 2018, date of current version June 20, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2841020

On Watermarking for Collaborative Model-Driven Engineering

SALVADOR MARTÍNEZ[1], SÉBASTIEN GÉRARD[1], AND JORDI CABOT[2]
[1]CEA-List, 91120 Paris-Saclay, France
[2]ICREA-UOC, 08860 Barcelona, Spain

Corresponding author: Salvador Martínez (salvador.martinez@cea.fr)

This work was supported by the Spanish Government under Project TIN2016-75944-R and the Electronic Component Systems for European Leadership Joint Undertaking under Grant 737494.

ABSTRACT Collaborative development scenarios often require models to be shared among the different stakeholders. These stakeholders are mostly remote with communication typically taking place over untrusted networks. This raises the need for effective intellectual property (IP) protection mechanisms for the shared models. Watermarking, an information hiding technique aimed at providing the means to verify the authenticity, integrity, and ownership of digital assets, has proved useful to provide IP protection in both media (images, audio, and video) and non-media domains (databases, XML documents, and graphs). In this paper, we explore the adaptation of the concept of watermarking to the modeling domain. We provide a novel and robust labeling mechanism based in the use of locality sensitive hashing and error correction codes. This labeling mechanism enables the integration of state-of-the-art watermarking algorithms in model-driven development approaches. Additionally, we leverage on the labeling mechanism to contribute a zero-watermarking algorithm to watermark models without introducing distortions to the data. We demonstrate the feasibility of our approach by providing a prototype implementation of our contribution.

INDEX TERMS Model-driven engineering, watermarking, intellectual property protection.

I. INTRODUCTION

Model-driven engineering (MDE) is a software engineering approach that considers models as first-class citizens of the development process [1], [2]. Models can be used in all phases of the process and in a variety of scenarios including, for instance, early verification and testing or even (semi)automatic code generation.

The increased adoption of this paradigm [3], including collaborative scenarios, requires effective intellectual property (IP) protection mechanisms. Indeed, systems are more and more complex every day and often integrate IoT components, AI, Big data, and other heterogeneous subsystems. As a result, outsourcing parts of the system design process becomes a necessity. This is also becoming true for digital assets in what constitutes a paradigm shift towards the model-driven co-engineering of systems design. In such scenarios, a given model, finished or under construction, may be shared among different stakeholders over possibly untrusted channels. This could lead to the leak of IP, likely triggering reputation and/or economical losses.

Access-control mechanisms for MDE [4] prevent unauthorized users to access the models but they alone do not suffice to deal with the problems derived from the intentional or unintentional leaks of IP from authorized parties. A mechanism for detection and tracking of IP, providing prosecution evidence for legal purposes, is needed as well. In order to tackle this problem, we propose here the integration of *digital watermarking* techniques in the MDE development cycle.

Digital watermarking is an information hiding technique used to verify the authenticity, integrity and ownership of digital assets by means of the introduction of imperceptible marks [5]. This technique has been extensively exploited in the domains of digital images, video and audio for IP protection purposes [5], [6]. However, the approaches developed for those media domains, based on the existence of static and highly noise-tolerant information, can not be directly applied to other non-media domains, that are much less noise-tolerant and where data modifications stemming from normal usage are common. Thus, some specific techniques have been provided for the domain of non-media data [7]. Concretely, watermarking approaches have been provided for relational databases [8], [9], XML documents [10], [11] and graphs [12], among others.

The strategies presented in the aforementioned non-media watermarking approaches rely on two key assumptions: 1) the existence of identifiers to uniquely label the parts of the data (i.e., tuples in database relations, nodes or edges in graphs, etc) that may be subject to watermarking; and 2) the possibility of modifying some of the identified data. In practice, in order to deal with the first assumption they rely on the use of immutable identifiers (such as primary keys), external identifiers, or the availability of numerical data (to rely in the persistence of their most significant bits to obtain robustness against data modifications). As for the second assumption, they rely on the existence of numerical data as well, so that the watermark can be introduced in their least significant bits.

Unfortunately, models, seen as a set of typed model elements that contain a list of properties, operation signatures and relations, do not adapt well to the previous assumptions. This is so for three main reasons: first, identifiers, when they exist, do not enforce referential integrity nor characterize the model element data and structure, making them easy to attack (by modification or re-generation) without impacting the usability of the model; second, numerical data is typically scarce or even absent; and three, although certain application domains do allow the introduction of slightly arbitrary modifications in the model data, this is not the most common scenario due to the precise required semantics that models must satisfy. This paper enables the use of watermarking for IP protection on models by adapting existing watermarking approaches to deal with the above limitations. In particular, the paper contributes, first, a robust (i.e., resistant to data modification to a certain degree) labelling mechanism based in the use of locality sensitive hashing and error correction codes that uniquely identifies model elements with respect to (w.r.t.) their contents and position in the model structure. This labelling mechanism facilitates using in the MDE field existing state-of-the-art watermarking algorithms for non-media data. Secondly, the paper contributes a zero-watermarking algorithm that benefits from the previous labelling mechanism to introduce a watermarking algorithm for models that does not introduce distortions to the data.

This algorithm works by extracting a data pattern from the model that is then operated with the watermark data to obtain a key. In case of an IP conflict, this key, previously stored by a trusted authority, can be used to extract the watermark from a suspect model and allow stakeholders to determine its ownership without having to share the original model. By effectively integrating watermarking techniques in the MDE ecosystem, models can be safely contributed to collaborative projects as the intellectual property of the involved parties is protected.

Finally, in order to demonstrate the feasibility of our approach, we provide a prototype implementation of our labelling and watermarking algorithms together with an evaluation of their efficacity and robustness.

The rest of the paper is organized as follows. Section II introduces the basic watermarking concepts. It is followed by a discussion on their adaptation to model driven engineering in Section III. Section IV describes in detail our labelling strategy wheres Section V introduces our zero-watermarking algorithm. Analise's and evaluation of our labelling and watermarking approaches is provided in Section VI. We discuss different variations of these contributions and special application scenarios in Section VII. Section VIII discusses related work. Finally, we present conclusions and future work in Section IX.

II. PRELIMINARY CONCEPTS

This section introduces (non-media) watermarking systems and their properties. We adopt and generalize here the notation and definitions introduced in the foundational papers in this area [8] and [10].

A. BASIC DEFINITIONS AND ALGORITHMS

Definition 1: Watermarking scheme. A watermarking scheme is a quintuple $< M, \eta, \gamma, \omega, \tau >$ *and two functions,* $insert(M, m, \eta, \gamma, sk)$ *and* $extract(M', \eta, \gamma, \alpha, sk)$.

with:

- M*: the structured or semi-structured data to be watermarked*
- η*: number of elements in* M
- γ*: a gap parameter that determines the number of elements marked, being* $1/\gamma$ *the fraction of elements marked.*
- ω*: the actual number of watermarked elements* ($\omega \approx \eta/\gamma$)
- τ*: the minimum number of correctly watermarked elements found in a suspect* M *needed for a positive detection of the watermark.*
- α*: the significance level of the test for detecting the watermark.*
- sk*: a secret key.*
- $insert(M, m, \eta, \gamma, sk)$*: a function that embeds a message into the original host data* M.
- $extract(M', \eta, \gamma, \alpha, sk)$*: a function that determines if a variation of* M, M' *have been watermarked or not.*

Remark 1: Note that:

- *in our work* M *corresponds to the model to be watermarked. In other domains* M *may correspond to any other structured or semi-structured data such as XML documents or graphs.*
- *watermarking only* ω *elements has a double purpose: 1) minimize the modified data; and 2) make more difficult to remove the watermark, being the probability of modifying a watermarked element* $1/\gamma$.
- *a watermarking scheme is by nature probabilistic, i.e., there is a chance that the watermark can be found in not watermarked data. Thus,* α*, the significance level, is used to asses that the probability of that to happen is low enough to make the watermarking scheme trustful and thus, usable.* α *is therefore used to determine the value of* τ*, so that the probability of finding* τ *correctly watermarked elements is very low.*

Algorithm 1 Generic Watermarking Algorithm

Require:
M: input data
η: number of elements
sk: user secret key
γ: gap
message: watermark message
α: threshold criteria

```
1:  procedure InsertWatermark
2:      DivideData(M, η)
3:      for all element m in M do
4:          id ← m.id
5:          if Elegible(m, id, sk, γ) then
6:              Mark(m, message)
7:          end if
8:      end for
9:  end procedure
10: procedure ExtractWatermark
11:     detected ← 0
12:     DivideData(M, η)
13:     for all element m in M do
14:         id ← m.id
15:         if Elegible(m, id, sk, γ) then
16:             if Check(m, message) then
17:                 detected + +
18:             end if
19:         end if
20:     end for
21:     if Threshold(α, detected) then
22:         probable IP violation detected.
23:     end if
24: end procedure
```

The concepts in Definition 1 are referenced in the generic pseudocode algorithm we show in Algorithm 1. The algorithm takes as parameters: M, the data to be watermarked, a secret key sk, the gap γ, optionally, the *message* to be used to mark the data and finally, α. Note that τ is determined from α.

The *InsertWatermark* procedure, in charge of marking the data, works as follows. First, the input data M is divided into parts. This could be a natural division such as taking tuples in a database relation, elements tags in XML or classes and objects in MDE or any other grouping strategy. Then, each of the elements m in the input data M is evaluated to determine whether they are chosen to be watermarked or not. This is done by performing an operation that takes into account the gap, the secret key and the identification (id) of the element (line 4). Note that the purpose of using a secret key is to keep the selected elements secret (being the watermarking algorithm generally public). Then, once an element is selected, it is watermarked by introducing in the element data a bit (the algorithm could be extended to introduce n bits) from the watermark *message* to hide (line 5). Note that the message is optional as it could just be a random pattern of bits randomly generated by a secret seed.

Correspondingly, the *ExtractWatermark* procedure, in charge of detecting marks in data, works as follows. It uses the same data division and the same eligibility criteria as in *InsertWatermark*. Then, for each element, it verifies if the bits where the mark should have been inserted contain the right values by using the *Check* operation. If true, it increments the count of detected bits (*detected* variable). Finally, the *Threshold* operation is called (line 19) to determine if, in light of the number of detected marks and the α criterion, we can consider our watermark detected (the number τ of found bits required for watermark detection is determined by the desired significance level α).

It Is Important to Note the Fundamental Relevance of the id Parameter in These Two Procedures: It is used, together with other static parameters, to determine which elements are taken into account for watermark insertion and extraction (as not all elements are used). Therefore, for a successful usage of these procedures, element's *ids* must be either unmodifiable unless destroying the data or calculated from the data but resistant to some amount of data modification (considering that beyond that amount of modifications, the element may be considered to be a different element and thus, its *id* different as well).

B. WATERMARKING PROPERTIES.

We have introduced the concept of watermarking and how it is used to mark data and detect IP violations. Here we present four properties that characterize non-media watermarking schemes. We define them below.

Robustness: refers to the capacity of the watermarking method to resist host data distortions due to its normal manipulation. When data distortions are attacks with the intention of removing the watermark this property is referred as security, however, as the intention of the data manipulation can not be easily determined, we will refer to both as *robustness*.

Invisibility: is the property of the watermark to not be noticeable by users. When talking about digital media watermarking this refers to what can be perceived by the human senses (watermark schemes often rely in human perception limitations). For non-media data the invisibility is not as straightforward and depends of the intended use of the data.

Detectability: refers to the ability of detecting the watermark without false positives. As introduced in Definition 1, this is often a probabilistic process and not binary (yes/no).

Blindness: a watermarking method is blind when the original data is not needed to detect/extract the watermark. Blind watermark methods are generally preferred as the original data does not need to be shared for verification nor stored unmodified.

III. WATERMARKING IN MDE

This section discusses how the previously introduced concept of digital watermarking can be adapted to the MDE

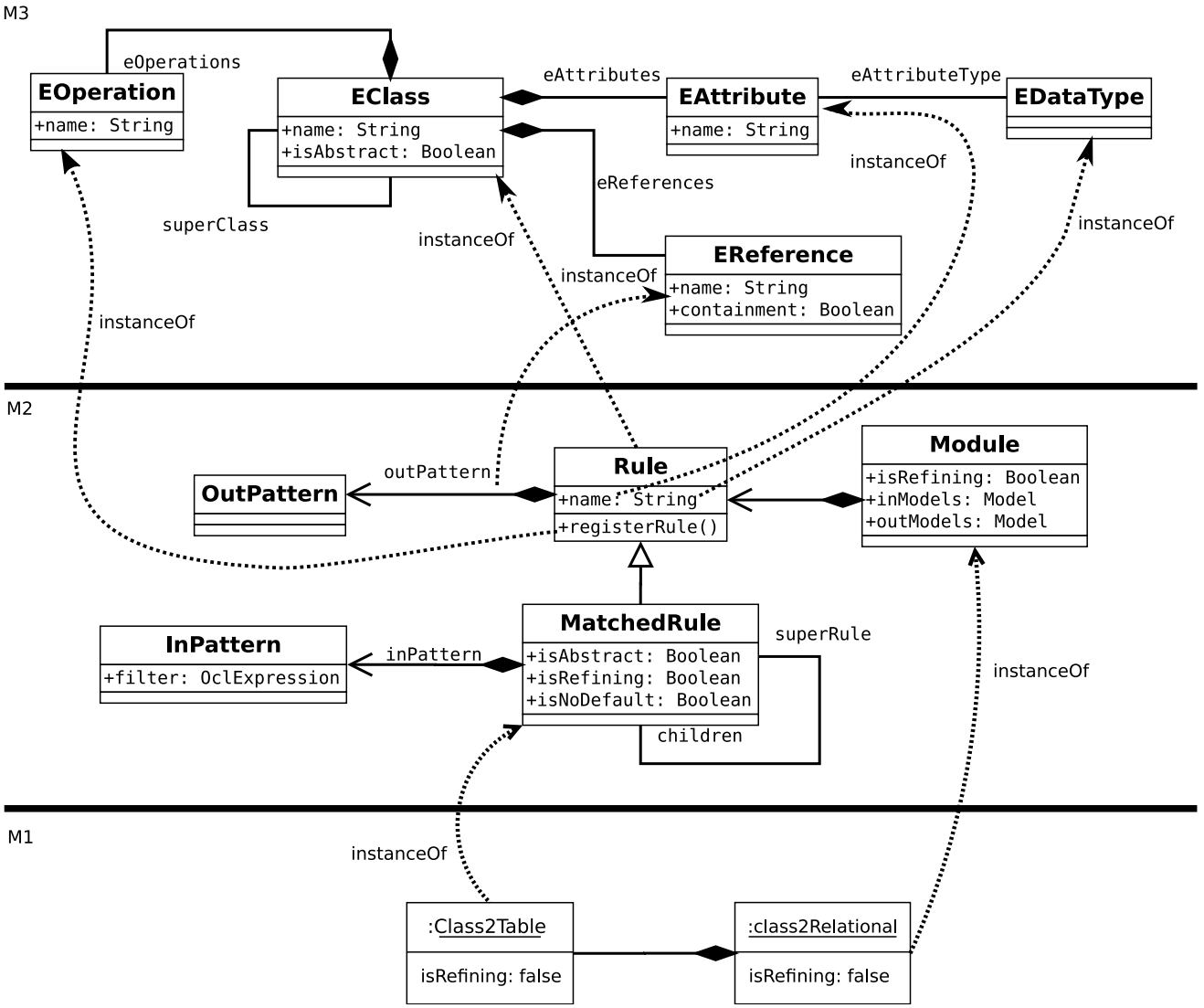

FIGURE 1. Models and metamodels.

ecosystem to enable the IP protection of models. In order to ease the discussion, we present in Figure 1 an example of real (i.e., used in academic and industrial scenarios) model.

The figure shows an excerpt of the conceptual model and abstract syntax of the ATL model transformation language. As it can be seen, a ATL transformation model (M1) is related to the ATL metamodel (M2) according to a relation of conformance (such relation is equivalent to the relation the code written in a given programming language has with respect to the grammar of that language). This metamodel provides the abstract syntax for the ATL language. It establishes that a transformation *Module* is composed by transformation *Rules* containing an *output pattern* (this is, a pattern to be created as the output of the rule execution). *Rules* are specialized into *Matched Rules* that include an *input pattern* (this is, a pattern to be matched to automatically trigger the rule execution). We write this ATL metamodel by combining classes, attributes (with their types), references and operations, concepts that are, in turn, defined by means of the third modeling level called metametamodel (M3). We show an excerpt of Ecore, the EMF [13] metamodeling language and de-facto standard, on top of Figure 1.

Summarizing, we define models as structured data (i.e., they conform to a given metamodel) composed of classes that contain a set of attributes, operation signatures and references (note that metamodels conform to metametamodels and, as such, they can be regarded (and manipulated) as models as well; we will use indistinctly the term model to refer to both unless disambiguation is necessary).

In particular, to create a watermarking system for MDE using the definitions and algorithms provided before we need to consider the following three questions:

What is the unit of data protection?

To answer this question we need to determine which information is *relevant* in a model and how we will handle models to fit them in the operations presented in Algorithm 1.

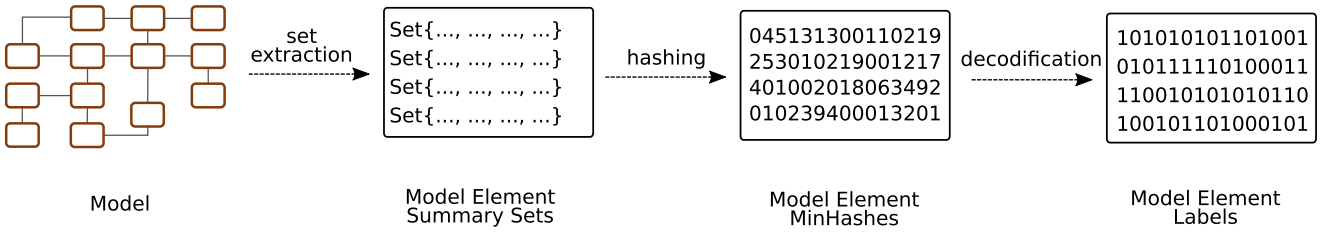

FIGURE 2. Model labelling process.

Following on the previous definition of the model concept, it is clear that classes (instances of EClass in M2) and objects (instances of class elements in M1) include both own data (attributes, operations) and are part of the model structure (via its references and relationships) and that both things need to be considered for IP protection. Then, our partition unit will be the model class/object. Smaller units (e.g. operations, attributes or references) do not contain much information alone and thus are not ideal candidates as partition unit.

How do we obtain the required ids?

We have already highlighted that, in order to use the procedures described in Algorithm 1, we need a labelling mechanism to uniquely identify model elements that is able to resist attacks (such as id removal or regeneration) and data modifications to a certain degree. Current id mechanisms in existing modeling frameworks do not fulfill these requirements. Thus, we provide a labelling mechanism for models in Section IV. Note that if a non-blind watermarking system is devised, it would be possible to use existing model-matching techniques to identify similar elements. However, this would come with the disadvantage of having to store (and compare) all versions of a given model, what would reduce the utility of the watermarking system.

How do we mark?

Although some domain specific models may tolerate modifications well, generally MDE artifacts have a very low bandwidth, this is, they tolerate a very low number of modifications. Therefore, to watermark in MDE we have that:

- for the case where models tolerate modifications, and assuming we have the required labelling mechanism as mentioned above, we can directly use the procedures presented in Algorithm 1 together with basic data watermarking techniques (e.g., numeric, text and categorical data watermarking).
- for the general case, a modification of those procedures, so that they do not introduce distortions in data but extract it called zero-watermarking is preferable. We provide the details of a zero-watermarking algorithm for models in Section V.

IV. A LABELLING MECHANISM FOR MODELS

As introduced in Section II, a non-media watermarking system for models requires a labelling mechanism, so that we can *uniquely* identify model elements in both, the watermark insertion and extraction procedures. Moreover, the labelling mechanism should not be easy to attack and work in a way that the labels do not change if the model element only changes slightly.

Figure 2 shows the overview of our proposed labelling mechanism for models. It consists of three steps. In the first step, summaries for each model element are generated. Then, summaries are transformed to numerical hashes with special properties. Finally, numerical hashes are decoded so that very similar model elements get the same label (*robustness* property).

We devote the rest of this section to the detailed description of the rationale and operationalization of each of step.

A. REPRESENTING MODEL ELEMENTS AS FEATURE SETS

One straightforward solution for labelling model elements would be to rely in persistent model element unique identifiers as provided by modeling frameworks like EMF [13]. Examples would be custom numerical IDs, UUIDs or even, when serialized in XMI, relative XPATH paths. Unfortunately, these static identifiers are very easy to attack as they do not depend on the model element content nor its context (only partially in the case of relative paths) enabling the attacker to easily change all the ids of the model without reducing its usability.

Alternatively, we propose the use of signature-based identities [14]. In this approach the identity of model elements is not determined by static identifiers but dynamically as an aggregation of its features. This way, identities depend on the model element content and its position in the model. We will call this signature-based identities *model summaries*.

Definition 2: Model Summary. Let M be a model element represented by the quintuple $< F, H, A, O, R, C >$ with:

- *F: the set of model inner features, such as name, id, abstractness, etc*
- *H: the set of super types.*
- *A: the set of attributes*
- *O: the set of operations*
- *R: the set of references*
- *C: the set of cross-references*

The summary of M, $SUMMARY(M)$ is the set $\{F \cup H \cup A \cup O \cup R \cup C\}$.

Remark 2: Note that in order to reduce the similarity between sets, we only use proper attributes of model elements,

discarding inherited ones. Note also that summaries can be tailored to specific domains, making the labelling system more resistant to domain specific modifications/attacks. As an example, we can create summaries for the domain of petri nets by considering places and transitions as model elements. Summaries for places and transitions could be sets containing their type, their labels, the incoming and outgoing arcs (with their weights) and in the case of places, the number of tokens.

B. HASHING FEATURE SETS PRESERVING THEIR SIMILARITY

Our goal in this step is to replace large feature sets by much smaller representations that we call *signatures*. In order for these signatures to be adequate for our labeling scheme they must present the following properties: 1) equal elements have the same signature; 2) similar elements have similar signatures, so that we can obtain, combined with the next step, resilience against modifications; 3) very different elements obtain very different signatures. Properties 1 and 3 are fulfilled by most hash functions (collisions being rare). Property two is trickier as for normal hashing functions hashes of similar values are very different due to the *avalanche effect*. In order to solve this issue, we adapt here the *min-wise independent permutations locality sensitive hashing scheme (minhash)*, used as an efficient way to detect near-duplicate elements in large datasets.

The *Jaccard Index* (see Definition 3) is an indicator of the similarity between two sets. The very basic idea of the minhash is that we can compare the minhash signatures of two sets and estimate the Jaccard similarity of the underlying sets from them alone. That is, the signatures preserve the *Jaccard Index* while being smaller and easier to manage than the original sets.

Definition 3: Jaccard Index. Let A and B be sets, the Jaccard similarity J(A,B) is defined to be the ratio of the number of elements of their intersection and the number of elements of their union:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Remark 3: J(A,B) is 1 when sets A and B are equal and 0 when A and B are disjoint.

Definition 4: Minhash Signature.

- *Let h_i with $0 < i < k$ be a collection of hash functions.*
- *Let S be a source set.*
- *Let $h_{min,i}(S)$ be the member x of S with the minimum value of $h_i(x)$.*

Then, the signature of S is the vector composed of all the $h_{min,i}(S)$ with $0 < i < k$:

$$SIGNATURE(S) = [h_{min,1}(S), h_{min,2}(S), \ldots, h_{min,k}(S)]$$

Remark 4: From [15] we know that for any i, $Pr[h_{min,i}(A) = h_{min,i}(B)] = J(A, B)$, this is, the probability of $h_{min,i}(A)$ to be equal to $h_{min,i}(B)$ corresponds to the original jaccard index *between A and B. It is demonstrated then that the average $h_{min,i}$ taken as binary variables is an unbiased estimator for the* jaccard index. *Thus, our signatures will preserve the* jaccard index *of the original summary sets (with some bounded error, inversely proportional to k).*

C. OBTAINING LABELS THROUGH ERROR CORRECTION CODES

Using *minhahs signatures* we have been able to compress our summary sets into numeric, smaller signatures that preserve the original similarity between sets. However, for our labelling system we need similar model elements (e.g., slightly modified classes) to get the exact same label, as we use these labels to decide which model classes are selected for watermarking. Usually, *minhash* signatures are used in a locality sensitive hashing scheme to detect near-duplicate elements by dividing the hashes in bands and classifying the bands into buckets so that good candidates for being duplicates can be discerned from bad candidates before doing the actual comparison with the original or source element. As in a blind watermarking scheme we do not have access to the original model, we cannot use this approach, instead, we adapt here the *fuzzy commitment* scheme originally introduced by [16].

Basically, the intuition behind a *fuzzy commitment* is that a safe containing a secret can be opened by using the secret key x but also by any secret key x' sufficiently near to x. This behavior is obtained by a unorthodox use of *error correction codes* that implies using the function that maps received messages to codewords directly on received secret keys without previously encoding them. This way, similar keys will be mapped to the same codeword (being the space of codewords smaller than the space of possible keys), thus opening the safe.

We borrow here the notation provided in [16] to *define error correction codes*.

Definition 5: Error correcting codes are a mechanism to add redundancy to messages to be transmitted over a noisy channel so that original messages can be recovered in the presence of errors. More formally and supposing, without loss of generality, binary messages:

Let $M \subseteq \{0, 1\}^k$ represent the space of messages, and $C \subseteq \{0, 1\}^n$ the space of codewords *(being codewords messages with added redundancy) error correction codes define two functions g and f such as:*

- *the function $g : M \rightarrow C$ translates messages M to codewords C while $g^{-1} : C \rightarrow M$ extracts messages from codewords.*
- *the function $f : \{0, 1\}^n \rightarrow C \cup \{\emptyset\}$ maps arbitrary n-bit strings to the nearest codeword (as stated in [16], nearest w.r.t. some distance measurement).*

Remark 5: Note that C contains 2^k codewords and $n > k$, so that redundancy can be added. If a code can correct up to t errors, the distance between codewords must be $2t + 1$. Finally, depending of the concrete error correction code used, f may return $\emptyset$ if it fails to find an unique corresponding codeword or a list of equally near codewords so that a selection mechanism must be in place.

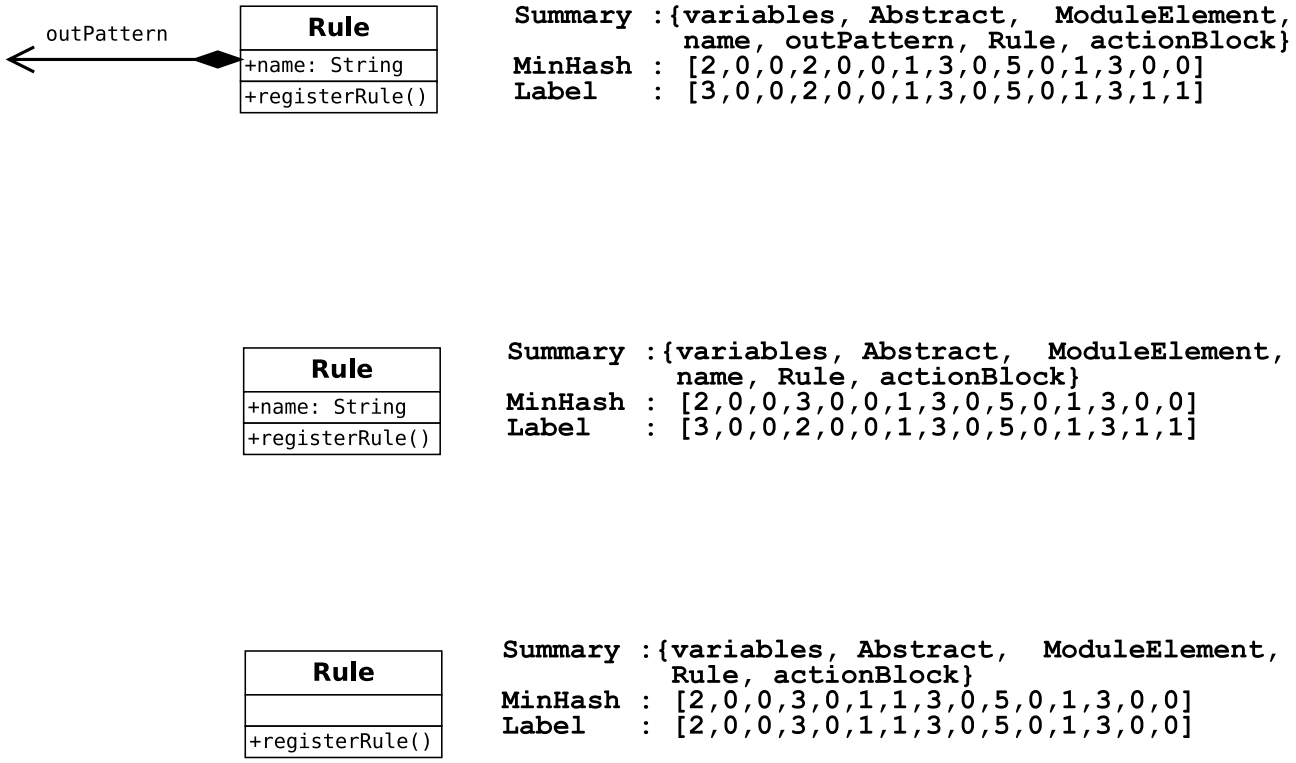

FIGURE 3. Labelling process example.

Following [16], our unorthodox use of error correction codes implies discarding the translation function g and using directly the mapping function f (it could be possible to use g^{-1} as well). In this setting, we will consider our *minhash signatures* as n-bit strings sent over a noisy channel, and ask f to give us the nearest codeword to it. This way, similar elements, with thus similar *minhashs signatures* will get the same codeword (if the difference between *minhashs signatures* is more than $2t + 1$, this is, there are more than t errors, we will get a different codeword as intended, we only need to choose a codeword space with an appropriate distance depending on the desired level of tolerance to modifications).

Algorithm 2 Labelling Algorithm

Input:
M: input model,
k: number of permutations for the minhash
sk: user secret key

1: **procedure** LABEL
2: INITMINHASH(k, sk)
3: **for all** model element m in M **do**
4: $summary \leftarrow$ SUMMARY(m)
5: $signature \leftarrow$ MINHASH($summary$)
6: $label[m] \leftarrow$ DECODE($signature$)
7: **end for**
8: **end procedure**

Algorithm 2 summarizes our labelling procedure. It takes as input the model to be labelled, the number of hash functions to be used in the calculation of minhash signatures and a secret key. It starts by calling the function *InitMinHash()* to create the k hash functions seeded with the secret key sk so that the labels can not be reproduced by an attacker. Then, for each model element in the input model M: 1) model element summaries are calculated by calling the function *SUMMARY()*; 2) each summary is used as an input to the *MinHash()* function, that returns a minhash signature for each of them; 3) finally, the operation *DECODE()* is called over each minhash signature so that we obtain the same label for very similar signatures. Note that this decode function could be any decode function of well known error correction codes such as Hamming codes or BCH codes (BCH codes would be more interesting as they can correct more than a single error and thus would make the labelling system more resilient to changes) and that the minhash signature can be considered as composed of several segments so that smaller error correction codes can be used.

Figure 3 illustrates how our labelling mechanism works on one of the classes of the example presented in Section II: 1) we start by labelling the *Rule* class following the Algorithm 2. We can see how the f function from the error correction code system assigns a codeword to the class that is different from the minhash signature; 2) as a second step, we mutate the *Rule* class by removing its relation to the *OutPattern* class. Then, re-calculate the label. We can see how the summary and

the minhash are affected by the mutation and end up differing from those of the unmutated class. However, the difference is corrected by the error correction code f function and we end up with the same label; 3) finally, we further mutate the *Rule* class by removing its name attribute and re/execute the label algorithm. We can see how the obtained summary and minhash signature are different from the previous class versions and how this time the error correction code f function does not map the minhash signature to the same codeword. The mutations made the class different enough from the original version as to consider it a different one.

V. A ZERO WATERMARKING APPROACH FOR MDE

Once we have a robust labelling mechanism, we can proceed to the proper watermark of models.

While a watermarking approach taking advantage of basic data watermarking techniques (e.g., the introduction of alterations in numeric, textual or categorical data) may be acceptable in some situations, modification of modeling data is generally not tolerable. Therefore, our watermarking approach should avoid data alterations.

Approaches able to insert a watermark without modifying the data exists for non-media domains (e.g., [17]). However, they are normally based on the introduction of a certain ordering on otherwise unordered elements (such as the order of tuples in a database relation) and thus are very fragile, as data can be randomly reordered by an attacker without affecting its utility. In MDE the situation is worsened as it is not uncommon to serialize a model in different ways (e.g, XMI, graph databases, etc) what could easily destroy such kind of ''sorting'' watermark.

As a consequence, and in order to provide MDE with and effective and practicable watermarking system we propose here the adoption of a zero watermarking approach for models. In a Zero-watermark approach, no alteration is introduced in the data. Instead, information is extracted from the model, manipulated and then stored by a trusted authority so that it can be later used in the case of IP violations.

Figure 4 summarizes the two basic steps required to zero-watermark a model: 1) the extraction of a pattern from a model; and 2) its operation with a given mask pattern to obtain a key representing the watermark.

Note that we do not use directly the pattern obtained from the model for security reasons. Indeed, as the *watermark* has to be stored by a trusted third party (or made public), it is desirable to reveal as less information as possible from the data we are trying to protect. This is achieved if the mask pattern employed to *xor* the extracted pattern is kept secret. We choose a *xor* cypher to *encrypt* the data for its simplicity. Alternatively, more secure encrypting mechanisms may be used if necessary.

We present in Listing 3 the adaptation to the zero watermarking of models of the *InsertWatermark* procedure presented in Section II, adapted to return the calculated watermark instead of inserting it (we renamed it here to *GetWatermarkKey* to make this change of functioning explicit).

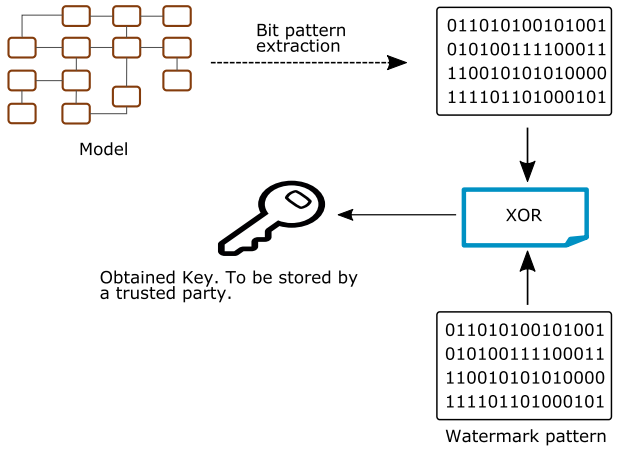

FIGURE 4. Watermarking approach.

It takes as an extra-parameter a number k, that establishes the number of permutations to be used by the minhash function of the labelling mechanism. The procedure starts by calculating the labels of all the model elements in the to-be watermarked model M (line 2). Then it initializes two vectors, *bitPattern* and *outKey*, used to store the extracted pattern from the model and the obtained watermark key respectively, a counter, *markedElements*, and a *hash table* storing all the labels. Next, the algorithm proceeds to select ω model elements to mark by using the hash table and the key *sk* (lines 7-8). Note that we use a hash table to store and retrieve the labels and corresponding model elements to avoid issues related with model traversal orderings influenced by concrete implementation or model mutations. The *mark* operation is substituted by the *GetNBits* operation in line 9, in charge of returning n bits from the given model element and append them to the *bitPattern* vector. Note that the origin of the n bits, this is, which of the elements participating in the summary set of the model is taken into account for the extraction, is decided using a random function seeded with the secret key *sk*. Finally, the watermark key is calculated in line 12 as a *xor* operation between the extracted bit pattern and the watermark message.

The *ExtractWatermark* procedure, that for brevity we do not show here, is, in our zero watermarking approach, almost equal to the *InsertWatermark* procedure. The only difference is the *message* parameter. In the extraction process it must be set to be the previously obtained *watermark key*. This way, the *xor* operation applied between the key and the bit pattern extracted from the model would yield, in case of an IP potential protection conflict:

1) the original message if it is an unmodified stolen model;
2) a message with some distortions but statistically sufficiently near to the original message to claim ownership if it is an stolen model with only some successful attacks.
3) a different message stating that the model does not contain the watermark if it is a different model or a model subjected to a successful attack (note that, as we

Algorithm 3 Watermark Generation Algorithm

Input:

M: input data,
sk: user secret key
γ: gap
message: watermark message
α: threshold criteria
k : number of permutations for the minHash

1: **procedure** GETWATERMARKKEY
2: LABEL(M, k, sk)
3: $outKey \leftarrow []$
4: $bitPattern \leftarrow []$
5: $markedElements \leftarrow 0$
6: $hashTable \leftarrow$ LABELSHASHTABLE(M)
7: **while** $markedElements < (M.size \div \gamma)$ **do**
8: $m \leftarrow$ GETFROMHASH(sk)
9: $bitPattern.append$(GETNBITS(m,n,sk))
10: $markedElements++$
11: **end while**
12: $outKey \leftarrow message \oplus bitPattern$
13: **end procedure**

show in Section VI, a successful attack requires an important number of modifications to the model, likely making it unusable).

VI. EVALUATION

We devote this section to the analysis and experimental evaluation of MDE watermarking approach. The fundamental watermarking properties presented in Section II, this is, *blindness*, *invisibility*, *detectability* and *robustness*, need to be analysed w.r.t. our watermarking approach to see how well it fares against them. For each property we first explain how they can be calculated and evaluated and then we present (when appropriate) the actual validation using a prototype implementation of our approach.

Our prototype (available online)[1] includes an implementation of our labeling mechanism and the zero watermarking algorithm. It has been implemented by using Java and the EMF API [13] for the calculation of model element summaries, minhashs signatures, watermarking algorithms and manipulation of Ecore models. We have delegated the implementation of the error correction codes to the *Octave* communications toolbox.[2] As for the scalability of the approach, minhash and error correction code algorithms have been conceived to work fast with massive amounts of data, so they scale very well. Scalability of MDE tools has improved in the later years and solutions exist to deal with very large models [18].

A. BLINDNESS

Our approach is *blind* as it does not require the original model for watermark extraction. While the extracted watermark need to be stored by a third party for later comparisons this is not different from other (non-zero) watermarking approaches were the watermark or the algorithm used to generate it needs to be conserved (as well as the secret keys used in the algorithms). All in all, our approach does not expose to the public more than other approaches, as the original model can no be reconstructed from an encrypted watermark.

B. INVISIBILITY

As we have chosen a zero-watermarking approach, this is, a watermarking approach that does not introduce any distortion to the data to be protected, our approach is, by nature, invisible. Other watermarking approaches introducing distortions may be built on top of our labelling mechanism. For those approaches the invisibility property will mainly depend on the method used to introduce the watermark in the host data and would require a specific analysis.

C. DETECTABILITY

Once we have our labelling mechanism in place, dropping thus the need for primary keys or stable numeric values, our watermark detection process can be modeled as in [8]. Concretely, this approach assumes that the watermark insertion/calculation consists in setting bit values according to independent tosses of a fair coin (as with a XOR operation with an independent mask). Then, supposing we look at ω bits to detect our watermark, the probability that at least τ bits matched the supposed assigned value is given by the cumulative binomial probability equation 1, where 0.5 is the probability of a bit being the expected one:

$$\sum_{i=\tau}^{\omega} \binom{\omega}{i} 0.5^i * 0.5^{\omega-i} \tag{1}$$

Note that, normally, we will expect to find $w/2$ matching bits in not watermarked models, and thus we will need to find more than 50% of correctly marked elements to asses watermark detection. Concretely, given a significance level α the threshold would be given by the minimum *tau* in equation 1 that yields a probability smaller than α.

From [8] and [10] we have also that the detectability of the watermark with a high level of confidence depends on the number of *marked* (in our case, extracted bits) elements and the watermark ratio $1/\gamma$. The more this two values grow, the less correctly *marked* elements are needed for a positive watermark detection with high confidence. Our zero-watermarking approach does not introduce any distortion in the model, and thus allows us to increase the number of taken bits as needed. Note however that there is a trade-off between detectability and robustness, as the more marked elements we have, the more chances that an attack modifies the watermark (note that this could be useful in a fragile watermarking setting, used to detect tampering instead of IP violation).

We have theoretically discussed how our approach and implementation performs regarding false positives. We are interested in verifying that this theoretical model holds.

[1] https://gitlab.com/smartine/mde-watermarking
[2] https://octave.sourceforge.io/communications/index.html

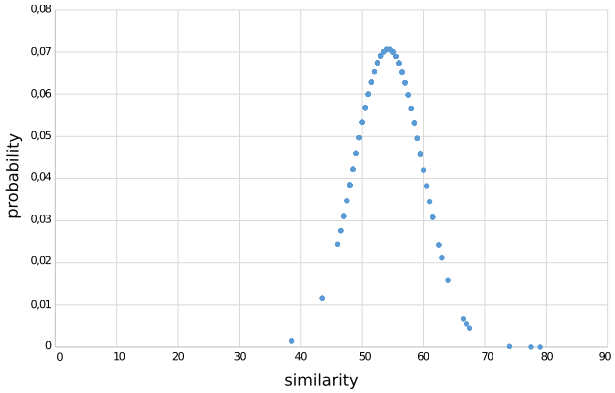

FIGURE 5. Similarity distribution.

This is, that in average two bit patterns extracted from two different models would have only around 50% of their bits in common.

Detectability (Experimental Validation): To conduct this evaluation we have: 1) selected 20 different metamodels from the ATL metamodel zoo[3]; 2) extracted a bit pattern of 200 bits from each one of them; and 3) calculated the bit pattern similarity of all bit pattern pairs (we have 190 different pairs). We use for that the simple matching coefficient (SMC), a statistic used to measure the similarity and diversity of sample sets seen as binary vectors and calculated as in equation 2, were M_{11} and M_{00} represent the number of elements with same values (either 0 or 1) for the sets under comparison and M_{01}, M_{10} the number of elements with different values.

$$SMC = \frac{M_{00} + M_{11}}{M_{00} + M_{01} + M_{10} + M_{11}} \quad (2)$$

We show in Figure 5 the distribution of the obtained similarity values (with the x-axis representing similarity and the y-axis probability) with mean equal to 54,23 and standard deviation equal to 5.64. As we can see, there is a small deviation w.r.t. to the theoretical model, with the most values grouping around the 55%-60% of similarity. This is due to two main causes: first, the use of default types (such as Boolean and String); and second, the existence of a relatively common *vocabulary* (e.g., it is very typical to define attributes named 'name' of type String).

The Figure 5 also shows three pairs with a higher similarity (more than 70). They correspond to three different pairs (sbvrEclipse.ecore-ifc2x3.ecore with 74 of similarity coefficient, Matlab.ecore-Maude.ecore with 77 of similarity coefficient and UML2.ecore-ProMarte.ecore with 79 of similarity coefficient).

The similarity between UML and ProMarte is easy to explain as both are modeling languages that reuse many concepts between them. Matlab and Maude both represent *programming* languages using similar vocabulary and structures (e.g., they both define Type and Parameter classes). SVBR and IFC represent very different domains (business rules and CAD model exchange) but use very similar concepts such as *Rule*, *Constraint* and *Actor*, what explain a higher than average similarity.

These similarity values, though higher than average, are still far from the majority of similarity coefficients (above 85) obtained when comparing the patterns extracted from a model and its mutations as we can see in Table 1 and thus they do not lead to false positives.

D. ROBUSTNESS

Once we have shown that our watermarking approach is not prone to false positives and thus, that the detectability property is as expected in the theoretical analysis provided above, we proceed here to discuss its robustness.

1) ROBUSTNESS AGAINST MODEL ELEMENT CONTENT ATTACKS

This version of the Robustness property refers to the resistance of model elements to modifications of its inner features (such as name, abstractness, etc) along with its attributes and operations signatures. Resistance to attacks modifying those values is obtained by two factors: 1) the watermark ratio $1/\gamma$, that decreases the probability of modifying a element used for the watermark and 2) the labelling mechanism, that itself is capable of resisting changes and yield the same label for slightly different model elements (note that only some bits are taken from each model element and that the origin of the bits, i.e., which of the attributes provides the bits, is decided randomly, and thus, being m the number of bits in a model element, the probability of modifying the right bit is $1/m$).

2) ROBUSTNESS AGAINST STRUCTURE ATTACKS

This property refers to attacks affecting the structure of the model, i.e. the insertion/deletion of model elements and relations. The impact of the addition of model elements is generally low as the only possible risk is that this new model element is taken into account for watermarking, being the possibility of that happening $1/\gamma$. The same applies to the deletion of model elements. If we delete a model element used before for watermarking (again with $1/\gamma$ chance of success), we will get n different bits as another model element would be taken its place. Many bits need to be deleted to succeed in the deletion of the watermark and at the cost of greatly reducing the usability of the model and therefore making the attack rather useless. The modification of relations would follow the same rationale as the modification of inner features, attributes and operation signatures, as being part of the model summary, they get resilience from the watermark ration and the labelling mechanism.

3) ROBUSTNESS: EXPERIMENTAL VALIDATION

To conduct the evaluation of the robustness property we perform the labelling and watermarking of three different regular-size models (our implementation works with Ecore models, but we also deal with UML models and Profiles). Each model is mutated (we use and adaptation of the

[3]http://web.emn.fr/x-info/atlanmod/index.php?title=Ecore)

TABLE 1. Evaluation results.

Mut.	Model	Survived Labels	Gap $\gamma = 2$	Gap $\gamma = 4$	Gap $\gamma = 10$	Gap $\gamma = 16$
10%	UML2	59%	smc=0.985	smc=1	smc=0.985	smc=0.985
25%	UML2	52%	smc=0.965	smc=0.955	smc=0.975	smc=0.95
50%	UML2	42%	smc=0.965	smc=0.955	smc=0.98	smc=0.94
75%	UML2	37%	smc=0.96	smc=0.94	smc=0.91	smc=0.75
10%	ATL	50%	smc=0.99	smc=1	smc:1	smc:1
25%	ATL	66%	smc=0.725	smc=1	smc:1	smc:1
50%	ATL	50%	smc=0.95	smc=0.95	smc:0.88	smc:0.85
75%	ATL	27%	smc=0.835	smc=0.945	smc:1	smc:1
10%	MARTE	50%	smc=0.985	smc=0.975	smc:1	smc:0.995
25%	MARTE	57%	smc=0.905	smc=0.905	smc:0.96	smc:0.875
50%	MARTE	41%	smc=0.94	smc=0.885	smc:0.85	smc:0.77
75%	MARTE	37%	smc=0.93	smc=0.93	smc:0.96	smc:0.885

EcoreMutator tool)[4] with several variation degrees to validate how well our approach is able to detect that the original model and the mutated one are a derivation from each other. The watermarking we employ in all cases consists in a pattern of 200 bits by using different gaps γ. Note that using average size models is not a limitation of the evaluation. On the contrary, it aims to show that our approach works in realistic scenarios, instead of only in extreme cases where the watermark is easier to "hide" in a very large model.

We show the results of our evaluation in Table 1. The first column details the number of mutations introduced in the model w.r.t. their number of elements. The second column identifies the model being tested (we have used the UML2 metamodel, counting 240 model elements, the MARTE profile, counting 116 model elements and the ATL metamodel, counting 28 model elements, all of them taken from the ATL model zoo). Third column shows the percentage of labels that survived to the mutation of the model element they identify. Fourth to seventh columns show the results of watermarking insertion and detection by using different gaps γ. These results show the similarity between the extracted mark and the expected one.

Regarding the labeling process, from Table 1 we can conclude that our labels resist mutation well and degrade softly as the number of introduced mutations increases. This is normal, as an increased number of mutations implies more mutations affect the same model element, distorting it beyond the point of recognition. Note that a single mutation can succeed to modify a label if the model element is not dense, this is, it does not have attributes, nor relations (explaining the results in the 10% mutation row). As for watermarking, detectability and robustness are verified as in most cases we can recover the watermark even in the presence of mutations with a high level of significance (being the probability of finding the number of found bits very low). In many cases, we have been able to recover the exact watermark (cells with smc $= 1$).

VII. VARIATIONS OF THE APPROACH

As stated in [19], watermarking mechanisms may have different potential applications such as transaction tracking, proof of ownership, copy control and authentication. We have explored the use of watermarking to provide proof of ownership in MDE. However, our labelling and watermarking approach can be easily modified to better suit different application scenarios (or to simplify its implementation). In the following we list a few of this variations/applications not limited to the ones mentioned by [19].

- **Non-blind (or informed) systems** implies storing the original to-be protected model so that it is available later in order to compare it with other models suspected of being derived from it. A typical use case could involve plagiarism detection in educative environments where the source model can remain available. While this would permit to use model comparators instead of our labelling system, for efficiency reasons it would be still recommended to use our labelling mechanism and compare the generated hashes instead of the models themselves. It would be interesting to test the robustness of the model comparators w.r.t. our labelling system.
- **Tampering detection** could be achieved by making our approach *fragile* instead of *robust*. This could be achieved by reducing the gap γ (so that more elements are used when extracting the bit pattern from the model), eliminating the error correction codes, and augmenting the total number of extracted bits. This way, a model modification would be easily detected.
- **Fingerprinting** models consist in embedding different watermarks in models shipped to different clients so that it is possible to detect the source of a leak. This may be easily done with our approach by using different secret keys.
- **Robust Hashing.** Robust hashing is a hashing technique aimed to obtain hashes of data that represent families of similar data instead of individual information. It have been explored in media and non-media domains: e.g., [20] defines robust hashing function for images while [21] do it for 3d models that unlike images may be modified often. An approach for text documents is described in [22].

  Our watermarking approach assumes the use of a robust labelling mechanism to choose model classes to either insert some modification or extract some information bits to create a model hash. If instead of extracting information bits we employ directly our robust labels to create the hash (different combination and aggregation

[4]https://code.google.com/archive/a/eclipselabs.org/p/ecore-mutator

approaches could be explored) we would obtain a robust hashing scheme, where the hash of the model would represent not only the current model, but a full family of similar models.

VIII. RELATED WORK

Digital data watermarking has attired a great deal of attention from the research community over the last decade. To the best of our knowledge, our approach is the first watermarking system specifically designed to the IP protection in the software modeling domain.

Agrawal's seminar paper [8] on watermarking relational databases constitutes the basis of most of the approaches later developed for relational data, XML documents or any other structured or semi-structured non-media data. Our approach also takes inspiration from it but drops two of its strongest limitations: our watermarking approach does not require the existence of numerical data in the model elements nor requires the existence of primary keys. In the XML domain, Agrawal's approach has been adapted in [11] to deal with compressed XMLs. It requires however that the user provides *locators*, this is, external identifiers to substitute primary keys. Zhou *et al.* [23] propose an approach based on the use of identifier queries, i.e., queries used to identify fragments of data. Their purpose is to obtain an approach resilient to changes in the organization of data. As a disadvantage, identifier queries need to be rewritten so that they keep working when data is reorganized. Still within the XML domain, Gross-Amblard [24] presents a theoretical work studying how to watermark databases and XML documents while preventing the output values of certain queries to change. Note that while it is common to store the models as XML files, our approach is generic and works no matter the specific storage mechanisms employed to persist the models.

More similar to our work, Sion *et al.* [12] present an approach to watermark graphs that protect both, the content of the graph and its structure. They propose a labeling mechanism for graph nodes. It is a complex iterative process that requires the prior mutation of the source graph to calculate label ranges. This way labels resist graph mutation attacks as long as they fall between the calculated range. Similarly, [10] provides a labelling mechanism for XML documents based of the assignment of weights to the different nodes in the path to the root. The labeling system presents some resilience to changes but only when the changes are limited to leaf nodes, being vulnerable to changes in the structure of the XML tree and therefore, less robust than our approach.

References [12] and [23] are "ad-hoc" labelling mechanisms. They need to be re-built with human intervention for each new asset to protect. In [12] mutations need to be defined and executed on a given model to obtain the label ranges while in [23] usability queries need to be identified for each asset and then re-written in case of attacks.

Although the mechanisms in [12] and [23] may be eventually more resistant than our method for a specific asset and attack, they require much more manual work and will only work well as long as the IP expert is able to correctly foresee all the potential attacks. Instead, our approach is automatic and protects against expected and unforeseen attacks. Moreover, none of these approaches specify how to obtain labels for non-numerical data (the most common type of data in models) and thus are not directly usable for MDE artefacts.

Finally, some software/code-oriented watermarking techniques may be considered similar to the watermarking of models (or graphs) since they insert the watermark in the control flow graph of a program [25]. They rely in the introduction of watermark sub-graphs linked to the control flow graph. Identification of watermark nodes in such a setting is done manually by inserting watermark node identification labels, which could be easy to identify and remove. We believe our approach could complement those approaches as our labels are dynamically calculated and more difficult to attack. In the same sense, 3D model watermarking could be assimilated to a kind of graph watermarking as usually 3D models are represented as graphs containing nodes, edges and facets [26]. However, 3D model watermarking is not directly usable for our kind of models as the process of watermarking this kind of graphs is very domain specific: e.g., it exploits geometrical information, the conservation of some perceptual properties is a constraint, and they are designed to stand some very specific attacks (e.g., changes in the model resolution, etc).

The utility of zero-watermarking as a watermarking mechanism that avoids altering data has been acknowledged by several approaches targeting different non-media data such as relational databases [27], [28], text documents [29], and XML [30]. However, the assumptions and techniques used in those works do not adapt well to the MDE domain. Our watermark generation algorithm is similar to the zero watermarking approach for audio proposed in [31] as we perform a XOR with a binary pattern (image) to obtain a watermark key. Our labelling mechanism and pattern extraction is however completely different and adapted to MDE instead to the audio domain.

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we have explored the adaptation of the watermarking concept to the software modeling domain as a key tool for IP protection purposes. IP protection is required in any non-trivial collaborative development scenario involving remote collaboration, specially in offshoring / outsourcing context.

We have shown how a novel labelling mechanism for models enables the use of state-of-the-art watermarking algorithms in MDE and demonstrated how it can be used in a zero-watermarking system were no alteration is introduced to the models. Besides, we have provided a prototype implementation of our approach that we have used to verify the important properties of detectability and robustness. As future work we intend to extend the present work by exploring four different research lines.

Concretely, we are interested in:

1) exploring the personalization of our approach to specific types of models, such as models representing executable code. The additional semantics of a specific model type can be used to improve labelling of models of that type.
2) extending our approach to protect sets of related models in what is typically known as a megamodel [32].
3) substituting the third party trusted authority with an alternative approach based on decentralized blockchain-based approach.
4) and, on a teaching setting, the possibility of using our labelling mechanism to find near duplicate models for automatic classification and plagiarism detection purposes.

REFERENCES

[1] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice* (Synthesis Lectures on Software Engineering), vol. 1. San Rafael, CA, USA: Morgan & Claypool, 2012, pp. 1–182.
[2] A. R. da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Comput. Lang., Syst. Struct.*, vol. 43, pp. 139–155, Oct. 2015.
[3] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE Softw.*, vol. 31, no. 3, pp. 79–85, May 2014.
[4] G. Bergmann, C. Debreceni, I. Ráth, and D. Varró, "Query-based access control for secure collaborative modeling using bidirectional transformations," in *Proc. ACM/IEEE 19th Int. Conf. Model Driven Eng. Lang. Syst.*, 2016, pp. 351–361.
[5] I. J. Cox, M. L. Miller, J. A. Bloom, J. Fridrich, and T. Kalker, *Digital Watermarking and Steganography*. San Mateo, CA, USA: Morgan Kaufmann, 2007.
[6] I. J. Cox, M. L. Miller, J. A. Bloom, and C. Honsinger, *Digital Watermarking*. Berlin, Germany: Springer, 2002.
[7] A. S. Panah, R. Van Schyndel, T. Sellis, and E. Bertino, "On the properties of non-media digital watermarking: A review of state of the art techniques," *IEEE Access*, vol. 4, pp. 2670–2704, 2016.
[8] R. Agrawal and J. Kiernan, "Watermarking relational databases," in *Proc. 28th Int. Conf. Very Large Data Bases (VLDB Endowment)*, 2002, pp. 155–166.
[9] R. Sion, M. Atallah, and S. Prabhakar, "Rights protection for relational data," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 12, pp. 1509–1525, Dec. 2004.
[10] G. Chen, K. Chen, T. Hu, and J. Dong, "Watermarking abstract tree-structured data," in *Proc. WAIM*. Berlin, Germany: Springer, 2005, pp. 221–232.
[11] W. Ng and H.-L. Lau, "Effective approaches for watermarking XML data," in *Proc. DASFAA*, 2005, pp. 68–80.
[12] R. Sion, M. Atallah, and S. Prabhakar, "Resilient information hiding for abstract semi-structures," in *Proc. IWDW*. Berlin, Germany: Springer, 2003, pp. 141–153.
[13] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2009.
[14] R. Reddy, R. France, S. Ghosh, F. Fleurey, and B. Baudry, "Model composition—A signature-based approach," in *Proc. Aspect Oriented Modeling (AOM) Workshop*, 2005, pp. 1–7.
[15] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. IEEE Compress. Complex. Sequences*, Jun. 1997, pp. 21–29.
[16] A. Juels and M. Wattenberg, "A fuzzy commitment scheme," in *Proc. 6th ACM Conf. Comput. Commun. Secur.*, 1999, pp. 28–36.
[17] S. Bhattacharya and A. Cortesi, "A distortion free watermark framework for relational databases," in *Proc. ICSOFT*, vol. 2, 2009, pp. 229–234.
[18] G. Daniel *et al.*, "NeoEMF: A multi-database model persistence framework for very large models," *Sci. Comput. Program.*, vol. 149, pp. 9–14, Dec. 2017.
[19] I. J. Cox and M. L. Miller, "The first 50 years of electronic watermarking," *EURASIP J. Adv. Signal Process.*, vol. 2, no. 2, p. 820936, 2002.
[20] J. Fridrich and M. Goljan, "Robust hash functions for digital watermarking," in *Proc. Int. Conf. IEEE Inf. Technol., Coding Comput.*, Mar. 2000, pp. 178–183.
[21] S.-H. Lee and K.-R. Kwon, "Robust 3D mesh model hashing based on feature object," *Digit. Signal Process.*, vol. 22, no. 5, pp. 744–759, 2012.
[22] M. Steinebach, P. Klöckner, N. Reimers, D. Wienand, and P. Wolf, "Robust hash algorithms for text," in *Communications and Multimedia Security*. Berlin, Germany: Springer, 2013, pp. 135–144.
[23] X. Zhou, H. Pang, and K.-L. Tan, "Query-based watermarking for XML data," in *Proc. 2nd ACM Symp. Inf., Comput. Commun. Secur.*, 2007, pp. 253–264.
[24] D. Gross-Amblard, "Query-preserving watermarking of relational databases and XML documents," in *Proc. 22nd ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, 2003, pp. 191–201.
[25] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation—Tools for software protection," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 735–746, Aug. 2002.
[26] K. Wang, G. Lavoué, F. Denis, and A. Baskurt, "A comprehensive survey on three-dimensional mesh watermarking," *IEEE Trans. Multimedia*, vol. 10, no. 8, pp. 1513–1527, Dec. 2008.
[27] Y. Jian, Z. Hongjun, H. Wenning, C. Gang, and L. Bin, "A zero-watermarking algorithm for relational database copyright protection," in *Proc. IEEE 3rd Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Jun. 2012, pp. 28–31.
[28] A. Khan and S. A. Husain, "A fragile zero watermarking scheme to detect and characterize malicious modifications in database relations," *Sci. World J.*, vol. 2013, Apr. 2013, Art. no. 796726.
[29] Z. Jalil, A. M. Mirza, and T. Iqbal, "A zero-watermarking algorithm for text documents based on structural components," in *Proc. Int. Conf. IEEE Inf. Emerg. Technol. (ICIET)*, Jun. 2010, pp. 1–5.
[30] Q. Wen, Y. Wang, and P. Li, "Two zero-watermark methods for XML documents," *J. Real-Time Image Process.*, vol. 14, no. 1, pp. 183–192, 2016.
[31] N. Chen and J. Zhu, "A robust zero-watermarking algorithm for audio," *EURASIP J. Adv. Signal Process.*, vol. 2008, p. 453580, Dec. 2007.
[32] J. Bézivin, F. Jouault, and P. Valduriez, "On the need for megamodels," in *Proc. OOPSLA/GPCE, Best Practices Model-Driven Softw. Develop. Workshop, 19th Annu. ACM Conf. Object-Oriented Program., Syst., Lang., Appl.*, 2004.

SALVADOR MARTÍNEZ received the Ph.D. degree in reverse engineering of security policies from the École des Mines de Nantes, France, in 2014. He was an Associate Professor with the École des Mines de Nantes. He is currently a Post-Doctoral Researcher with the CEA-LIST LISE Laboratory in collaboration with the SOM Research Group, Universitat Oberta de Catalunya. He works on the integration of security concerns on the artefacts of the model-driven environment. He has also a research expertise in core model-driven engineering technologies, such as the efficient execution of model transformations and in software evolution with a focus on security concerns (concretely, in access-control policies). He has been involved in the European OPEES (ITEA) and MONDO (FP7) projects.

SÉBASTIEN GÉRARD received the degree in mechanical and aeronautics from the Superior School of Mechanics and Aeronautics, Poitiers, France, in 1995, and the Ph.D. degree in computer science in 2000. He is currently a CEA LIST Senior Researcher in software engineering and computer science. He is also leading a research team of about 20 engineers at CEA LIST (an arm of the French Atomic Energy Agency) within the Laboratory for Model-Based Engineering of Real-Time and Embedded (RT/E) systems. The principal objective of this research of this team is to achieve correct-by-construction design of RT/E systems from requirements to implementation. Through his involvement in a numerous national and international research projects, he has worked with many industrial partners, such as Peugeot Citroen, Airbus, ST Microelectronics, EADS, gaining extensive experience, and insight into industrial problems and requirements. He is also deeply involved in various standardization activities, and is currently co-chairing both the UML 2 and MARTE (the UML extension for RT/E) standardization task forces. He is also a core member of the European network of excellence, ArtistDesign, where he is an expert on issues related to modeling and standardization.

JORDI CABOT received the B.Sc. and Ph.D. degrees in computer science from the Technical University of Catalonia. He was a Leader of an INRIA and LINA Research Group, École des Mines de Nantes, France, a Post-Doctoral Fellow with the University of Toronto, a Senior Lecturer with the Open University of Catalonia (UOC), and a Visiting Scholar with the Politecnico di Milano. He is currently an ICREA Research Professor with the Internet Interdisciplinary Institute, Universitat Oberta de Catalunya. His research interests include software and systems modeling, model-driven and web engineering, formal verification and social aspects of software engineering, topics on which he has published over 150 peer-reviewed conference and journal papers. He is currently leading the SOM Research Group, UOC. He is the PI of the National Spanish Project Open data for all and leads the team participation in the ECSEL EU Project MegaMart2 and represents the UOC in international networks and consortiums, including the Eclipse Polarsys Group and the Papyrus Industrial Consortium. Apart from his scientific publications in international conferences and journals in these areas, he writes and blogs about all these topics in his modeling languages portal.