Robust Hashing for Models

Salvador Martínez
CEA List
Paris-Saclay, France
salvador.martinez@cea.fr

Sébastien Gérard
CEA List
Paris-Saclay, France
sebastien.gerard@cea.fr

Jordi Cabot
ICREA - UOC
Barcelona, Spain
jordi.cabot@icrea.cat

ABSTRACT

The increased adoption of model-driven engineering (MDE) in complex industrial environments highlights the value of a company's modeling artefacts. As such, any MDE ecosystem must provide mechanisms to both, protect, and take full advantage of these valuable assets.

In this sense, we explore the adaptation of the *Robust Hashing* technique to the MDE domain. Indeed, robust hashing algorithms (i.e. hashing algorithms that generate similar outputs from similar input data), have been proved useful as a key building block in intellectual property protection, authenticity assessment and fast comparison and retrieval solutions for different application domains. We present a novel robust hashing mechanism for models based on the use of model fragmentation and locality sensitive hashing. We discuss the usefulness of this technique on a number of scenarios and its feasibility by providing a prototype implementation and corresponding experimental evaluation.

CCS CONCEPTS

• **Software and its engineering** → **Model-driven software engineering**; **System modeling languages**; • **Information systems** → *Similarity measures*; *Near-duplicate and plagiarism detection*;

KEYWORDS

Robust Hashing, Near-duplicate and plagiarism detection, Locality Sensitive Hashing

ACM Reference Format:
Salvador Martínez, Sébastien Gérard, and Jordi Cabot. 2018. Robust Hashing for Models. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3239372.3239405

MODELS '18, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-4949-9/18/10…$15.00
https://doi.org/10.1145/3239372.3239405

1 INTRODUCTION

The increased adoption of model-driven engineering (MDE) tools and techniques in complex industrial environments (e.g., MDE for cyber-physical systems, Internet of Things or Industry 4.0.) highlights modeling artefacts as valuable assets. Indeed, in such complex scenarios, valuable (domain-specific) knowledge is produced, exchanged and consumed in the form of models and metamodels, transformations and queries. Consequently, new requirements for MDE arise while existing ones get accentuated.

Concretely, in order to both, protect, and take full advantage of these valuable assets, we believe that the MDE ecosystem needs to provide the means to 1) protect the MDE asset's owners from losing their intellectual property 2) protect MDEs asset's users from malicious tampering aimed at damaging or putting at risk the assets 3) efficiently store, retrieve and compare MDE assets, so that reuse is maximized.

In order to deal with the aforementioned requirements, we explore in this paper the adaptation of the concept of *robust hashing* to the MDE domain. Indeed, robust hashing algorithms have been proved useful as a key building block for providing intellectual property protection, authenticity assessment and fast comparison and retrieval solutions in different application domains such as digital images [14], 3D models [20] or text documents [34]. Contrary to cryptographic hash algorithms, such as MD5 [29] or SHA1 [11], where slightly different inputs produce very different outputs due the avalanche effect [13], robust hashing algorithms (often called perceptual hashing algorithms) produce the same or very similar hashes for similar inputs. Moreover, they are capable of resisting attacks (i.e., modifications) that change non-essential properties of the asset.

Therefore, we propose in this work a novel robust hashing algorithm for MDE artefacts. Our algorithm starts by extracting from the model to be hashed multiple overlapping fragments, so that model elements are characterized not only by their contents (e.g., attributes and operations) but also by their relative position w.r.t. other model elements. Fragments are then translated to textual set summaries so that we can apply to them the *min-wise independent permutations locality sensitive hashing scheme (minhash)* [5], that reduces large summary sets to small and robust hash signatures. Finally hash signatures are manipulated and combined to obtain the final robust hash of a given model.

We demonstrate the feasibility of our approach by a prototype implementation for EMF models and its corresponding evaluation w.r.t. the properties of robustness (i.e., resistance to mutations and thus to false negatives) and discrimination (absence of false positives).

The rest of the paper is organized as follows. Section 2 introduces the basic concepts of a robust hashing schema and Section 3 their adaptation to MDE. Section 4 provides the details about our hashing algorithms and its building blocks. Analisys and evaluation of our robust hashing approach are provided in Section 5, followed by a discussion of possible application scenarios in Section 6. Section 7 discusses related work. Finally, we present conclusions and future work in Section 8.

2 BACKGROUND ON ROBUST HASHING

We introduce here the basic concepts and properties related to the robust hashing technique.

2.1 Robust Hashing

Classical cryptographic hashes such as SHA1 or MD5 may be used for the authentication and integrity assessment of digital assets. However, and due to the avalanche effect they include in their design, small changes to the asset lead to the generation of very different hashes, making them unsuitable for other tasks such as fast comparison and retrieval, intellectual property protection or plagiarism detection. This is so because in these scenarios we are interested not only in finding exact assets, but also variations of the assets.

In order to solve this problem, the concept of robust (or perceptual) hashing has been introduced, notably in the domain of digital images [14] but also in other domains such as those of 3D mesh models [20] and textual documents [34]. A robust hash is a hash that can resist a certain type and/or number of data manipulations. This is, the hash obtained from a digital asset and that from another asset similar to the original one but that has been subjected to minor manipulations should be the same or at least very similar. As an example, robust hash algorithms for images or 3D model mesh resist manipulations such as rotation and compression, as they remain visually *similar*. Text documents remain similar if they convey the same message, thus, they resist to attacks introducing synonyms, etc.

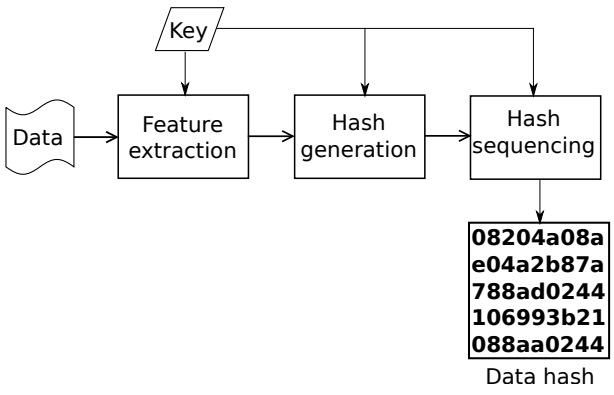


Figure 1: Robust Hashing Generation Algorithm

The main building blocks of a robust hashing schema are depicted in Figure 1. Basically, a robust hash algorithm starts by extracting key features from the data to hash, then groups and hashes them to finally use aggregation/selection/compression methods to obtain the final data hash. We describe each of these buildings blocks in the following. Note that these building blocks may be decomposed in several substeps.

- *Feature extraction.* Core features of the data to be protected are extracted (e.g., for images this could be histogram information, wavelet transform information, etc.). The key idea of this feature extraction step is to focus in those characteristics that give the data its main *meaning*, so that changes in less important features do not lead to very different hashes. This step normally divides the input data in several different data groups, then applies a randomization step to scramble the data groups.
- *Hash Generation.* The core features extracted in the previous step are then further manipulated and finally hashed using different mechanisms (that can include cryptographic hashes). As with the previous step, randomization operations are applied to the results to further scramble the hash.
- *Hash Sequencing.* The hashes generated from features in the previous step are transformed into a final hash. This step may include different operations with different objectives such as compressing the hash or augmenting its robustness. Among the typical operations that we find in this block we have: 1) quantization; and 2) error correction codes.
- *Key.* Unlike standard cryptographic hash algorithms, robust hash algorithms are by construction vulnerable to *pre-image attacks*, notably to *second pre-image* attacks. Knowing the hash algorithm, it is not difficult to find an input that leads to a given hash. And given a data input and a hash it is not difficult to build a second input with the same or a very similar hash. To alleviate this problem, the previously described steps use randomization operations. Being that we need the randomization to be *reproducible* so that we can do hash comparisons, a secret key is used as a seed to the pseudo-random number generator used at various stages of the robust hashing algorithms. Given the same data, if a different key is used, the resulting hash will be statistically independent and thus completely different. The key is kept secret, and so, the hash value of a given piece of data cannot be computed or verified by an unauthorized party.

2.2 Robust Hashing Properties And Requirements

We have introduced the concept of robust hashing and its composing building blocks. Here we present two properties that characterize robust hashing algorithms: *Robustness* and *Discrimination*. We define them below:

- *Robustness* refers to the capacity of a robust hashing algorithm to resist to data distortions due to 1) its normal manipulation; or 2) intentional attacks aimed at hiding the similarity between the data. In other words, it refers to the capacity of the robust hash algorithm not to produce false negatives in the face of certain data manipulations that do not change the core features of the data.
- *Discrimination* refers to the ability of a robust hashing algorithm to tell the difference between two different, not related (i.e., not derived from one another) data inputs. This is, a robust hash algorithm, for being usable, needs to avoid producing false positives.

Summarizing, when designing a robust hash algorithm, both robustness and discrimination need to be assured. There is a trade-off between both properties: the more robust, the more prone to false positives and viceversa.

As extra requirements for a robust hashing scheme we have the size of the hash and its internal organization. Hashes must have a regular size and their features must be scrambled following a predictable algorithm. Otherwise, the hash comparison for determining authenticity of the data will not be feasible (calculating similarity and distance of irregular data is much more difficult and often extremely slow). Moreover, note that for certain applications, the *detection* of similar hashes will be a probabilistic process. This is, we may not find exact the same hash but a *very* similar one. Thus, a threshold criteria would have to be defined to discriminate between similar and non similar hashes.

3 REQUIREMENTS FOR A ROBUST HASHING SOLUTION FOR MODELS

This section discusses how the previously introduced concept of robust hashing and its related properties can be adapted to the MDE ecosystem to enable the hashing of models. Next sections describes how we take these requirements into account in our proposal for a robust hashing algorithm for models.

In an abstract way, models can be regarded as structured data (defined by the metamodel the model conforms to) composed of model elements that contain a set of attributes and/or reference slots (other models may be then built by using this basic building blocks). Similarly, 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. And in fact, all MDE artefacts can be represented as models themselves and thus, will benefit from our approach [2].

In this scenario, and based on the definitions from the previous section, we need to answer the two following questions in order to design a robust hashing algorithm for models: 1) what is the information of a model that is essential?; 2) what are the model modifications or attacks that need to be resisted by our algorithm?

3.1 Essential features

Model elements include individual data (attributes, operations) but are also related to a number of other model elements via their references. Both aspects need to be considered for a robust hashing. If we consider only the content, two models with the same elements would generate the same hash even if those elements were organized according to a very different structure. And, similarly, if we just take the structure into account, models of two very different domains but that, by chance, share a similar structure, could be regarded as equivalent.

3.2 Type of attacks

A model can be modified in very different ways. Some modifications should not have any impact in the resulting hash. Some other modifications should be tolerated to a certain degree. Concretely:

- Changes in the way a model is serialized should not have any impact on the resulting hash. The same model stored as an XMI document, as table rows in a relational database or as a graph object in a graph database should always result in the same hash.
- Mutations to the model content, e.g., for UML structural models, changes to classes, attributes, references, operations, etc., should be resisted but only to a certain degree. A structural model containing a few new classes and references should be still considered the same (or a derived) model (as per the robustness property). This resistance should not be that strong that models that become different enough to be perceived as so by a human are classified as equal (as for the discrimination property).

4 APPROACH

This sections describes our proposal for a robust hashing mechanism for models. Figure 2 depicts its main steps. It starts by creating m fragments of size s from the model to hash. These model fragments are then represented as text summaries so that we can apply to them the *minhash* hashing technique to obtain m small model fragment signatures. Once the model signatures have been generated, the next step consists in dividing and *re-hashing* them to perform a content-based classification in a number of buckets. Next, the global model hash is created by taking n elements from key-selected buckets. Finally, the hash is compressed by using a scalar quantization step.

We devote the rest of this section to a detailed description of the rationale and functioning of each of these steps. Note that in order to ease the discussion we use as a concrete target for the process UML structural models. Nevertheless, our approach is designed to work for any kind of model.

4.1 Model Fragmentation

As a first step we create a number of fragments from the model. This allows to generate the hashes using as unit not the single model element (which, as we have discussed before,

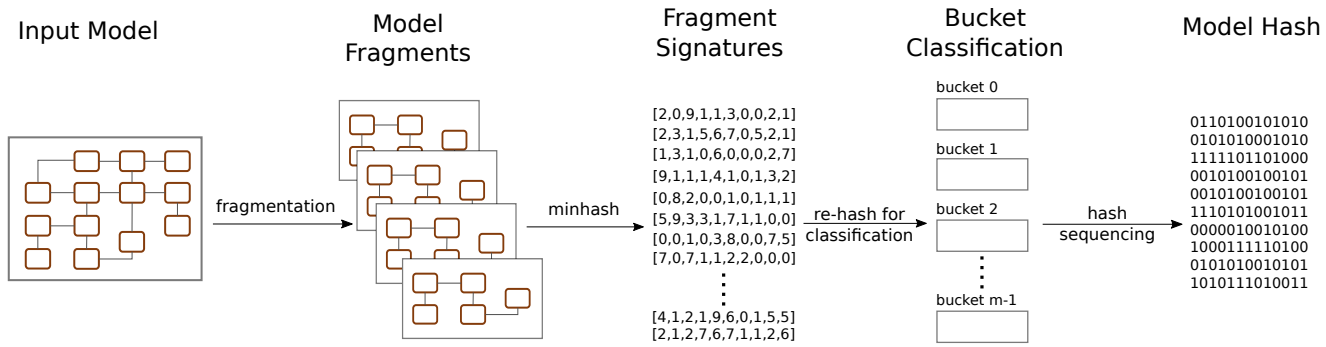


Figure 2: Model Hashing Process

would not be good enough) but the element together with some contextual information.

Once we have determined the need for the extraction of model fragments we need to decide: 1) how our fragments are created and 2) how many of these fragments are needed. The answer to these two questions is determined by the purpose of our method, that is to resist model mutations (including the addition and deletion of model elements).

The principal requirement our fragment needs to meet is its statistical independence from the other fragments, so that a change affecting one fragment would not be propagated to other fragments creating the avalanche effect we precisely need to avoid.

Advanced model slicing techniques, aiming at extracting a subset from a model for a specific purpose (monitoring, model comprehension, modularity,...) [18] [4] [36] may be adapted to our use case. However, as we do not require our fragments to fulfill any *semantic* criteria, we propose a simpler approach, closer to approaches that automatically generate model fragments for the purposes of test generation[6] or efficient model storage [30].

[6] and [30] produce disjoint fragments. We need to drop this constraint so that model fragments are created in an independent way. In the same sense, we need to extract *many* fragments and not just a small number of them. In Definition 1, we show how we build fragments around a model element acting as *center*. The process of selecting a small number of centers would be greatly affected by mutations by imposing different orders to the list of existing models elements. We avoid this problem by extracting a large number of fragments (experiments show that the best results are obtained when the number of fragments approximates to the number of model elements in the model) and by later using content-based classification to select just a few of these fragments for the final hash. We will see in Subsection 4.3 how the classification step guarantees that 1) we take fragments that are different, maximizing the coverage of the model to hash; 2) we take similar fragments even in the presence of mutations.

Definition 1. *Connected Model Fragment. Let A be the set of all model elements in a given model, $c \in A$ the model element center of the fragment and C^* the closure of neighbours of c (being neighbours any element reachable from c through its references, including the superclass relation), then, we describe a fragment of A with center in c:*

$$F_c^n(A) = \{x_i : x_i \in A, x_i \in C^*, 0<i<n\}$$

Remark. *Note that in order to avoid ordering issues related to the modeling API traversal functions and/or model mutations we first store model elements in a hash map so that the selection of fragments centers is as consistent among executions as possible.*

4.2 Signatures of Model Fragments

Previously to the hashing of the extracted model fragments, we need to transform them to *summary sets*, this is, sets containing words representing the contents of the fragment. This enables us to then use the *minhash* technique to obtain the fragment signatures. We use content-based [28] identities of model elements as the base for the translation from model fragments to summary sets. We call this content-based identities *model summaries* and *model fragment summaries*.

Definition 2. *Model Element Summary. Let M be a model element represented by the quadruple $< F, H, A, O >$ 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*

We have $SUMMARY(M) = \{F \cup H \cup A \cup O\}$.

Remark. *Note that in order to reduce the similarity between sets, we only use proper attributes of model elements, discarding inherited ones. Note also that we provide here a very generic summary, aimed to work at the abstract syntax level of any model. However, summaries can be modified for some specific domains, so that other information is added (e.g., constraints) and/or different weights are given to the constituting parts of model elements, making the labelling system more or less resistant to better fit the need of specific scenarios. In this same sense, different summaries could be*

conceived for different modeling domains such as petri nets or sequence diagrams.

Definition 3. *Model Fragment Summary. Let $F_c^n(A)$ be a model fragment composed of n model elements M_i, we have*

$$SUMMARY(F_c^n(A)) = \bigcup_{i=1}^{n} SUMMARY(M_i)$$

Remark. *Relations between elements are implicitly captured by the concept of connected fragment. This is, all elements in a fragment are connected with one or another kind of relation. Relations may be added explicitly to the fragments (or the individual model fragments). Note however that adding many details of the relation, such as cardinalities, etc., will increase the discrimination capacity of the approach but decrease its robustness.*

Once we have our fragments represented as summary sets, our goal is to replace them with much smaller representations that we call *signatures*. In order for these signatures to be adequate for our robust hashing scheme, they must be robust themselves, this is: 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. 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 4) is an indicator of the similarity between two sets.

Definition 4. *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. *J(A,B) is 1 when sets A and B are equal and 0 when A and B are disjoint.*

The virtue of the minhash is that we can compare 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.

Basically, a signature for a given number of sets is built by: 1), creating an ordered dictionary of existing words for the universe of sets to hash; 2), creating a characteristics matrix by assigning 1 if the word exists in the set to hash and 0 if it does not exist. As an example, being the dictionary {this, is, a, dictionary} and two sentences to be hashed: 'a dictionary' and 'this dictionary', the characteristics matrix would be [0,1;0,0;1,0;1,1]; finally, the hashes are constructed by permuting this matrix and selecting, for each sentence to hash, the minimum row index with a non-zero value. The signature will have as many components as permutations. Having the original matrix and the permutations [1,0;0,0;0,1;1,1], [1,0;1,1;0,1;0,0] and [1,1;0,0;1,0;0,1], the signature for the 'a dictionary' will be [2,2,2,3] and for 'this dictionary' [0,0,3,3].

However, as permutations are very expensive to implement, they are usually simulated by the use of random hash functions. We adopt here this latter approach. More formally, hash-based minhash signatures are defined as follows:

Definition 5. *Minhash Signature.*

- *Let h_i with $0 < i < k$ be a collection of hash functions and S 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. *From [5] we know that for any i, the following equality holds. $Pr[h_{min,i}(A) = h_{min,i}(B)] = J(A, B)$. 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).*

We show in Figure 3 an example of the two first steps of our approach. Concretely, we show a random fragment extracted from the ProMarte.ecore metamodel obtained from the ATL Metamodel Zoo[1]. The fragment has its *center* in the *TimingMechanism* class and includes four neighbours (*Clock*, *TimedEvent*, *MetricTimeValue* and *MetricTimeInterval*).

Below the fragment diagram we show the corresponding *summary* and *signature*. Then, we show the summary and signature of the fragment after a mutation that removes the *Clock* class. Finally, we show the summary and signature of a totally different fragment.

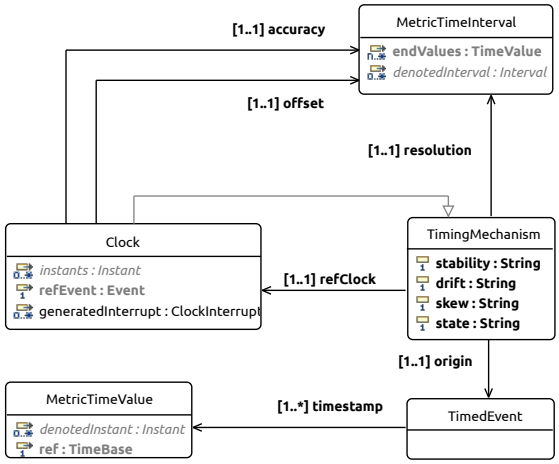
From this example we can see that similar fragments (e.g., a fragment after a mutation) get very similar signatures (only positions six and fourteen of the signatures are different) while different fragments get totally different signatures.

Note that as per Definition 2, model summaries only include proper attributes, discarding reference names, cardinalities, etc. This is not a limitation of our approach as, effectively, model summaries could include other elements if needed to adapt them to specific application scenarios and *attacks*.

4.3 Classification of Model Fragments and Hash Sequencing

The three previous steps have allowed us to transform a model into a long set of minhash signatures. As explained before, we have decided to generate a large number of these signatures so to avoid model mutations to influence the hash

[1] http://web.emn.fr/x-info/atlanmod/index.php?title=Zoos



Fragment A Summary:
{TimingMechanism, TimedEvent, state, Clock, MetricTimeValue, stability, MetricTimeInterval, drift, skew}
Fragment A Minhash Signature:
[3, 1, 2, 3, 0, 6, 1, 2, 4, 5, 1, 2, 0, 10, 0, 0, 5, 0, 2, 5]

Mutated Fragment A Summary:
{TimingMechanism, TimedEvent, state, MetricTimeValue, stability, MetricTimeInterval, drift, skew}
Mutated Fragment A Minhash Signature:
[3, 1, 2, 3, 0, 9, 1, 2, 4, 5, 1, 2, 0, 13, 0, 0, 5, 0, 2, 5]

Fragment B Summary:
{NFPCategory, QualitativeNFP, NFPLibrary, AnnotatedModelElement, Quantity, NFP}
Fragment B Minhash Signature:
[5, 8, 5, 1, 17, 4, 11, 12, 1, 2, 15, 3, 1, 3, 1, 4, 0, 2, 2, 4]

Figure 3: Fragment Signatures

creation process (e.g., modifying the order of selected elements). However, much of the information generated in this manner is very redundant, and thus, can not be efficiently used as is for the generation of the model hash. Instead, the idea is to just take a certain number of different signatures by performing a classification step that puts similar signatures in the same bucket. Then we can just choose one of the elements of the bucket as its representative. Moreover, while model mutations may affect this selection step, the degree of the error would be minimized (if an element ends up in the same bucket, that means it is still quite similar) and not propagated across buckets.

This classification process is called *Locality Sensitive Hashing* (LSH). One general approach to LSH is to "hash" items several times, in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items are[31].

Having minhash signatures, an effective way to perform the hashings is to divide the signatures into b bands consisting of r rows each[21]. For each band, there is a hash function that takes vectors of r integers (the portion of one minhash signature within that band) and hashes them to some large number of buckets. We can use the same hash function for all the bands, but we use a separate bucket array for each band, in order to avoid inter-band collisions. Intuitively, the banding strategy makes similar minhash signatures much more likely to be candidate pairs than dissimilar pairs. We refer the reader to [21] for a more detailed description of how this band-classification increases the probabilities of similar signatures to end up in the same buckets.

Practically, this classification step is performed by:

(1) dividing the minhash signatures in bands (e.g., we could divide minhash signatures of 20 elements into 4 bands of 5 elements each). In effect, it is more likely that a given band remains equal after mutation than a whole hash. Thus, working with bands we add robustness to our method. Note however that bands should be sufficiently large so that false positives are unlikely.
(2) rehashing the selected band by using any available hash function. This could be achieved with java *hashcode* functions.
(3) applying a modulo operation on the number of buckets to the obtained hash to classify them in a given bucket.

Once the classification is done, the hash is constructed by polling the buckets randomly with the help of the secret key in order to take the desired number of minhash signatures (the same list of buckets will be selected across execution, assuring us to always take similar minhashes when we are dealing with similar models). Note that in case of selecting an empty bucket, the polling algorithm will just take a minhash signature from the next bucket.

4.4 Scalar Quantization For Compression

The last step of our robust hashing approach is optional. If reducing the size of the final hash and/or provide it with extra robustness to model changes is needed, the use of an scalar quantizer is advised (see Definition 6).

A scalar quantizer assigns each value of the obtained hash to a given interval so that the number of bits required to represent them gets reduced. The process will also make the final hash more resistant to mutations as very similar hashes would have values that will fall in the same interval. Alternatively, although arguably more complex, error corrections codes may be used to obtain a similar effect by exploiting the fuzzy commitment technique [15] consisting in directly using the decode function without the prior encoding of the data. This way, the hash data would be treated as a message received through a noise channel, then *decoded* to assign it the *nearest* codeword in the given error correction code.

Definition 6. *k-level quantizer.*

- *Let* d_i *with* $0 < i < k+1$ *be decision levels where* d_i *divides the range of data under quantization into* k *consecutive intervals* $[d_0, d_1)[d_1, d_2)...[d_{k-1}, d_k)$.
- *Let reconstruction levels* $r_0, r_1, ...r_k - 1$ *be the centers of the intervals.*
- *Let* v *be a value to quantize.*
- *Then,* $quantized(v) = i$ *when* $v > d_i$ *and* $v \leq d_{i+1}$

As an example of quantization, if we use twenty hash functions for the *minhash* signatures, we would obtain vectors of twenty elements with values from zero to nineteen, each of one requiring five bits for its representation. By creating intervals of just two consecutive values (e.g., we assign the values two and three to two) we will be able to represent each value with four bits, saving twenty bits for the whole minhash signature.

5 EVALUATION

We devote this section to the analysis and experimental evaluation of our robust hashing approach for models. The robust hashing properties presented in Section 2, this is, *discrimination*, and *robustness*, are analyzed to see how well our approach fares against them.

We first discuss for each property the expected behavior as derived from the construction of our approach. Then we present the actual validation using our prototype implementation. This prototype (available online[2]) includes the implementation of the model fragmentation, summary calculation, minhash signature generation and LSH (as band-based classification). It has been implemented using Java and the EMF API [33].

The hashing libraries integrated in our approach are highly performant. As such, the scalability of our approach is not an issue. As an example, it takes less than one second to produce the hash for even the larger models (around 1000 elements) in our experiments below on a fairly standard computer[3].

5.1 Discrimination

Discrimination is the ability of a robust hashing algorithm to produce different hashes for (very) different models. Effectively, while we want our robust hashing algorithm to resist modifications, it would be useless if it can not tell when two models are not related. We achieve a high level of discrimination by using connected model fragments as the basis for our feature extraction. Indeed, by working with fragments and not with isolated model elements, our approach is able to distinguish between models using similar vocabulary but with a different structure.

We validate this property by means of conducting the following empirical evaluation. We have: 1) selected 18 different metamodels from the ATL metamodel zoo; 2) hashed them with our robust hashing algorithm. We produce hashes of 200 bytes, by using 20 hash functions for the minhash signatures and by extracting 80 fragments from each model (we use 20 hash function to get signatures of 20 components so that collisions of different model fragments are very unlikely. The size of the final hash is then determined as a multiple of the size of the signatures and similar to the typical size of hashes in literature. Finally, the number of fragments is determined by experimental results showing that for the models used in the evaluation, more fragments did not improve robustness nor discrimination.) and 3) calculated the similarity of the hashes between all pairs.

We use for the calculation of the similarity the *Hamming Similarity* measure for vectors. It is calculated as shown in Equation 1. Basically, it counts the numbers of equal elements (*IdSim() returns 1 when elements are equal, 0 otherwise*) in the same position of the vectors and divides the result by the total number of elements.

$$HammingDistance(s,t) = \frac{\sum_{i=1}^{n} IdSim(s[i],t[i])}{n} \quad (1)$$

We show in Table 1 the result of the pairwise similarity calculation (multiplied by 100). The darker the cell, the more similar the hashes. As we can see, only the hashes obtained from the same model get higher levels of similarity, while the hashes of different models get similarities normally lower than 50. It is interesting to see that the higher similarity value between two different models corresponds to the pair UML2-J2SE5 that reach a similarity coefficient of 60. The UML2 and the J2SEE share vocabulary and structure (classes contain attributes and operations, have types, etc), and thus this high level of similarity is somehow expected. Nevertheless, the obtained similarity values obtained in this experiment are far from the similarity coefficients obtained when comparing the hashes obtained from a model and its mutations (see Table 2). Therefore, our robust hashing algorithm does not lead to false positives. We can thus conclude that the discrimination property holds for out robust hashing approach.

5.2 Robustness

Once we have shown that our robust hashing approach is not prone to false positives we proceed here to discuss its robustness, this is, its ability to resist modifications. As discussed in Section 2 our approach should resist to two types of modifications: 1) modifications related to the storage mechanism; and 2) modifications of the model content.

Our approach is robust against changes in the storage of the model as the storage information is not taken into account in the hashing process. As long as the implementation can provide model fragments as sets of model elements, our approach leads to the same results. In the same way, our approach is independent of the modeling framework used for the specification of a model (e.g., UML models, EMF models, GME [19],...). Note however that if required our approach could be sensitive to the specificities of a given modeling framework by including those specificities in the fragment summaries.

As for the modifications to the models contents and structure, our approach provides two protection mechanisms. First, the very use of *minhash* for the calculation of model fragment signatures gives robustness to our approach. This is illustrated in Figure 3 where a fragment and its mutated counterpart obtained very similar signatures. Second, the use of band-based *LSH* for the classification of signatures w.r.t. their content provides robustness to our approach as it promotes the selection of the same (or very similar) model

[2] https://gitlab.com/smartine/RobustModelHashing

[3] An Intel Core i5-6200U CPU @ 2.30GHz 4 cores, running ubuntu 16.04

Table 1: Pairwise Similarity

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
a	100	15	11	15	11	8	10	29	34	20	15	6	9	9	14	10	7	13
b	15	100	10	13	11	5	12	31	13	47	40	8	21	10	24	16	10	22
c	11	10	100	15	12	7	14	9	15	13	7	13	13	33	18	11	25	9
d	15	13	15	100	12	6	25	10	36	13	8	13	11	11	6	10	5	10
e	11	11	12	12	100	4	60	10	15	10	7	30	7	17	7	5	7	5
f	8	5	7	6	4	100	3	6	6	8	6	5	8	7	7	7	36	9
g	10	12	14	25	60	3	100	8	13	12	7	30	11	16	7	10	6	9
h	29	31	9	10	10	6	8	100	17	51	28	6	16	9	17	14	10	18
i	34	13	15	36	15	6	13	17	100	10	10	11	11	14	7	11	8	10
j	20	47	13	13	10	8	12	51	10	100	35	7	19	11	20	13	11	20
k	15	40	7	8	7	6	7	28	10	35	100	5	15	9	54	12	10	38
l	6	8	13	13	30	5	30	6	11	7	5	100	7	30	4	9	5	5
m	9	21	13	11	7	8	11	16	11	19	15	7	100	7	17	38	12	41
n	9	10	33	11	17	7	16	9	14	11	9	30	7	100	14	8	15	8
o	14	24	18	6	7	7	7	17	7	20	54	4	17	14	100	16	40	46
p	10	16	11	10	5	7	10	14	11	13	12	9	38	8	16	100	11	18
q	7	10	25	5	7	36	6	10	8	11	10	5	12	15	40	11	100	15
r	13	22	9	10	5	9	9	18	10	20	38	5	41	8	46	18	15	100

*a:SBVRvoc; b:mlhim2; c:Agate; d:sbvrEclipse; e:J2SE5; f:Matlab; g:UML2; h:SCADE; i:XHTML; j:KDM; k:Maude; l:MoDAF-AV; m:ifc2x3; n:MavenMaven; o:OpenConf.owl; p:ProMarte; q:MICRO.owl; r:SWRC;

Table 2: Mutation Resistance

	Number of Mutations			
Model Name	5 mt.	10 mt.	25 mt.	50 mt.
ProMarte	77	80	79	65
Uml2	79	96	62	76
Scade	100	80	74	34
OpenConf	77	81	93	58
Matlab	76	81	81	45

fragment signatures for the composition of the final model hash even in the presence of mutations.

To conduct the experimental evaluation of the robustness property we perform the hashing of five different regular-sized models (our prototype implementation works with Ecore models, but we also deal with UML models and Profiles). Then, we randomly introduce mutations to these five models. The introduced mutations include: adding and removing classes, adding and removing attributes and references, adding and removing generalizations and modifying attributes. (we use and adaptation of the EcoreMutator tool[4]). Finally, we compare the hashes of the original model to those of the mutated versions to determine whether their similarity level allow us to conclude that the models are derivations.

Table 2 summarizes the obtained results. Rows indicate the model, while columns indicate the number of introduced mutations (we introduce 5, 10, 25 and 50 mutations). Note that mutations are introduced randomly and independently between rows, i.e., we start always from the original model. This explains why models with five mutations sometimes get lower levels of similarity. From the results we can see that models resist very well the introduction of up to 25 mutations, with similarity values much higher than the ones obtained for different models as shown in Table 1. The last columns show how after 50 mutations models become too different to be considered similar. Notably Scade and Matlab, that are the smaller models (with around 100 and 34 models elements each), get the biggest impact.

As a summary, we can conclude that our robust hashing algorithm resists well mutations. Models need to be mutated to the extent of making them unusable for their original purpose in order to obtain very different hashes. Therefore our hashing algorithm is, as intended, robust.

5.3 Threats to Validity

The validity of the conclusions obtained by our experiments may be affected by:

(1) Our evaluation dataset: effectively, we focus our evaluation on metamodels. While we chose a large set of them, alleviating the threat to the validity of the conclusion within this kind of models, extending this conclusions to other kind of models will require further experimentation.

(2) The mutation generation: indeed, different kinds of mutations are randomly introduced in models. This way it is difficult to assure that a model with more mutations have been subject to more modifications than another with less but more severe mutations (e.g., a class deletion mutation affects a model more than an attribute modification mutation). Nevertheless, from the experimental evaluation we can conclude that the variations due to the random process are less important when the number of introduced mutations

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

grows (e.g., in some cases 5 mutations affect the model hash more than 10 mutations in 4 cases, while 10 mutations affect the model hash more than 25 only in one of the cases).

6 DISCUSSION OF APPLICATION SCENARIOS

A robust hashing algorithm for MDE artifacts makes it possible to efficiently support a number of application scenarios related to the fast search and classification of models, intellectual property protection and authenticity assessment. In this sense, in the following we briefly describe some of these applications.

6.1 Classification & Search

We can use the fast comparison and retrieval properties of a robust hashing schema to detect very similar models in large repositories (such as MDE Forge [1]). This could be useful in diverse scenarios, including: automatic classification of models (e.g., to find all models related to some topic); plagiarism detection in academic assignments, model diversity assessment, etc.

In order to do so, we need to extend our robust hashing algorithm with a *locality sensitive hashing* schema aimed at reducing the total amount of comparisons to complete when classifying a large number of models. Even if comparing hashes is much faster than comparing the models themselves, it is still computationally expensive. The goal is to first classify the models to compare into a set of buckets where each bucket holds models that are similar.

The process is very close to our classification step in Section 4, but this time applied to the whole model hash and not to fragments. As in that step, the idea behind this is that most of the dissimilar pairs will never hash to the same bucket (being the number of buckets large enough to avoid accidental collisions).

Once this is done, plagiarism detection gets simplified by reducing it to the problem of comparing the models that ended up in the same bucket. Conversely, model diversity could be ensured by choosing a set of models from different buckets (e.g. to increase coverage in a model-based testing scenario).

Within a bucket we can use standard model comparison tools to directly compare pairwise the models as the amount of comparisons is now drastically reduced. Indeed, LSH is an approximate search mechanism, and thus false positives while rare, may appear. Thus, model comparison and matching tools such as EMFCompare [7] DiffMerge [8], Epsilon Comparison Language [17] or [12] may be used to obtain more accurate results at this point.

6.2 Copyright Infringement

Robust hashing can be used to provide prosecution evidence in cases of IP protection violations. A model considered to be an unauthorized copy or derivation of a proprietary model would be hashed (using the owner's secret key) and this hash then searched and compared in order to determine its prior existence [5].

For this scenario, the importance of hashing relies not only on its efficient storage (as in the previous section) but also as a way to avoid exposing intellectual property as the models can not be reconstructed from the hashes (notably, without the secret key). Moreover, the fact that our hashing algorithm is robust is key to detect tampering aimed to avoid copyright infringement detection even for models are derivations from ours.

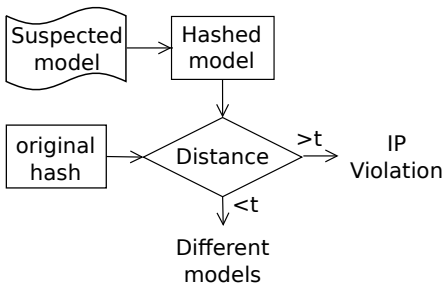


Figure 4: Ip Violation Detection Process

The detection of IP infringements follows the process depicted in Figure 4. Starting from a suspected model, we hash it by using our secret key. We then compare it with the hash of the original model to determine if the similarity of the hashes is high enough to determine, within a determined confidence level (this confidence is build as to make statistically very unlikely that such a similarity level arises from independent models), that they are either the same model or derived from each another. Note that in order to efficiently retrieve the original model from its storage, or to proactively find copies along repositories, we can use the classification and search mechanism described in Subsection 6.1.

6.3 Accountability

In a collaborative modeling scenario where a number of different companies participate in the creation and evolution of models, we typically want to keep track of the changes performed by each party and the corresponding model versions generated at each step. This information could be, for instance, stored in a blockchain infrastructure.

The benefits of the use of blockchain technologies as a decentralized consensus ledger to provide accountability for the actions of different agents on shared resources have been already acknowledged in different application domains [26] [24]. However, its adaptation to the MDE environment risks to present scalability issues. Indeed, directly storing full models as part of blockchain transactions would fail to scale as models are too large for what current blockchain infrastructure can efficiently handle nowadays.

[5]The hashing may also be used as a key building block in the construction of a watermarking schema [14] for models but to be most effective the hashing creation process should be different from the one presented here [23].

In this sense, our hashing mechanism for models may become a key enabler for the use of blockchain technologies for the enforcement of accountability in collaborative development scenarios by remarkably reducing the required storage requirements if we store the hash, instead of the full model, in the blockchain transactions. Similarly, looking for all transactions pertaining to a given model would be easier and quicker to perform by using the model digests provided by our hashing algorithm instead of their full model counterparts.

7 RELATED WORK

Robust hashing algorithms for different digital assets have 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 robust hashing algorithm specifically designed for MDE artefacts.

Notably many different robust hash algorithms have been proposed for the domains of digital images image [14] [16] [22] [37] [39], 3D mesh models [20] [38], digital video [9] [10] and text [34].

While the general, high-level process for generating the robust hashes is similar across domains, the concrete steps need to be adapted to each domain and to the desired robustness (e.g., which kind of modifications are to be resisted). We base our approach on the extraction of model fragments and its processing via the minhash [5] and locality sensitive hashing [21] methods for nearest neighbour search for text. The minhash technique has been used in [23] in order to provide a robust labelling mechanism for model elements that enables the use of state-of-the-art watermarking algorithms in the MDE ecosystem. While such watermarking algorithms could be used to extract patterns (instead of performing insertions) to be used as hashes, those hashes are less effective as model *digests* than the ones we generate here as they were created with a different and very specific goal in mind (e.g. those hashes are based in the extraction of some random bits to protect the watermark from being destroyed).

Prior to us, text analysis techniques for the purpose of model comparison have been used in [25], where the authors use Natural Language Processing (NLP) to find the semantic similarities between language descriptions by attaching text descriptions to domain concepts. We see their approach as complementary to ours, since these annotations could be added to the models before the hashing phase in order to take into account model semantics in our approach.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we have explored the adaptation of the robust hashing concept to the modeling domain. We have shown how the adaptation of the *minhash* and *locality sensitive hashing* techniques, developed for the comparison and classification of large repositories of text documents, can be used to effectively extract the core features of a model to produce a robust hash.

Our robust hashing algorithm for models is meant to become a key building block for providing solutions for intellectual property protection, authenticity assessment and fast comparison and retrieval, all of them key requirements for the adoption of model-driven engineering in complex industrial environments.

As future work we intend to extend the present work by exploring six different research lines. Concretely, we are interested in:

(1) comparing the similarity measures of our hashes with different existing model difference and similarity metrics [27] [35].
(2) exploring the personalization of our approach to specific types of models. The additional semantics of a specific model type can be used to improve the hashing of models of that type.
(3) extending our approach to the hashing of sets of interrelated models in what is typically known as a megamodel [3].
(4) integrating it in an effort to build a blockchain for models infrastructure, where transactions on models must be stored in en efficient way.
(5) extending our approach to enable the search of semantically meaningful fragments withing models (e.g., search for model patterns).
(6) exploring the introduction of machine learning techniques in order to automatically derive good hash function for specific model datasets [32].

REFERENCES

[1] Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Amleto Di Salle, Ludovico Iovino, and Alfonso Pierantonio. 2014. MDEForge: an Extensible Web-Based Modeling Platform.. In *CloudMDE@ MoDELS*. 66–75.

[2] Jean Bézivin. 2005. On the unification power of models. *Software & Systems Modeling* 4, 2 (2005), 171–188.

[3] Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. 2004. On the need for megamodels. In *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

[4] Arnaud Blouin, Benoît Combemale, Benoit Baudry, and Olivier Beaudoux. 2011. Modeling model slicers. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 62–76.

[5] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*. IEEE, 21–29.

[6] Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry, and Yves Le Traon. 2006. Metamodel-based test generation for model transformations: an algorithm and a tool. In *Software R eliability Engineering, 2006. ISSRE'06. 17th International Symposium on*. IEEE, 85–94.

[7] Cédric Brun and Alfonso Pierantonio. 2008. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional* 9, 2 (2008), 29–34.

[8] O Constant. [n. d.]. EMF Diff/Merge, 2012. ([n. d.]).

[9] Baris Coskun and Bulent Sankur. 2004. Robust video hash extraction. In *Signal Processing Conference, 2004 12th European*. IEEE, 2295–2298.

[10] Cedric De Roover, Christophe De Vleeschouwer, Frédéric Lefebvre, and Benoit Macq. 2005. Robust video hashing based on radial projections of key frames. *IEEE Transactions on Signal processing* 53, 10 (2005), 4020–4037.

[11] D Eastlake 3rd and Paul Jones. 2001. *US secure hash algorithm 1 (SHA1)*. Technical Report.

[12] Jean-Rémy Falleri, Marianne Huchard, Mathieu Lafourcade, and Clémentine Nebut. 2008. Metamodel matching for automatic model transformation generation. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 326–340.
[13] Horst Feistel. 1973. Cryptography and computer privacy. *Scientific american* 228, 5 (1973), 15–23.
[14] Jiri Fridrich and Miroslav Goljan. 2000. Robust hash functions for digital watermarking. In *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on*. IEEE, 178–183.
[15] Ari Juels and Martin Wattenberg. 1999. A fuzzy commitment scheme. In *Proceedings of the 6th ACM conference on Computer and communications security*. ACM, 28–36.
[16] Ram Kumar Karsh, RH Laskar, and Bhanu Bhai Richhariya. 2016. Robust image hashing using ring partition-PGNMF and local features. *SpringerPlus* 5, 1 (2016), 1995.
[17] Dimitrios S Kolovos. 2009. Establishing correspondences between models with the epsilon comparison language. In *European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 146–157.
[18] Kevin Lano and Shekoufeh Kolahdouz Rahimi. 2011. Slicing techniques for UML models. *Journal of Object Technology* 10, 11 (2011), 1–49.
[19] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. 2001. The generic modeling environment. In *Workshop on Intelligent Signal Processing, Budapest, Hungary*, Vol. 17. 1.
[20] Suk-Hwan Lee and Ki-Ryong Kwon. 2012. Robust 3D mesh model hashing based on feature object. *Digital Signal Processing* 22, 5 (2012), 744–759.
[21] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of massive datasets*. Cambridge university press.
[22] YuLing Liu and Yong Xiao. 2013. A robust image hashing algorithm resistant against geometrical attacks. *Radio Eng* 22, 4 (2013), 1072–1081.
[23] Salvador Martínez, Sébastien Gérard, and Jordi Cabot. 2018. On Watermarking for Collaborative Model-Driven Engineering. *IEEE Access* 6 (2018), 29715–29728.
[24] Ricardo Neisse, Gary Steri, and Igor Nai-Fovino. 2017. A blockchain-based approach for data accountability and provenance tracking. *arXiv preprint arXiv:1706.04507* (2017).
[25] Florian Noyrit, Sébastien Gérard, and François Terrier. 2013. Computer assisted integration of domain-specific modeling languages using text analysis techniques. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 505–521.
[26] Fernando Gomes Papi, Jomi Fred Hübner, and Maiquel de Brito. [n. d.]. Instrumenting Accountability in MAS with Blockchain. *Accountability and Responsibility in Multiagent Systems* ([n. d.]), 20.
[27] Nam H Pham, Hoan Anh Nguyen, Tung Thanh Nguyen, Jafar M Al-Kofahi, and Tien N Nguyen. 2009. Complete and accurate clone detection in graph-based models. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 276–286.
[28] Raghu Reddy, Robert France, Sudipto Ghosh, Franck Fleurey, and Benoit Baudry. 2005. Model composition-a signature-based approach. In *Aspect Oriented Modeling (AOM) Workshop*.
[29] Ronald Rivest. 1992. The MD5 message-digest algorithm. (1992).
[30] Markus Scheidgen. 2013. Reference representation techniques for large models. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*. ACM, 5.
[31] Malcolm Slaney and Michael Casey. 2008. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal processing magazine* 25, 2 (2008), 128–131.
[32] Jingkuan Song, Yi Yang, Xuelong Li, Zi Huang, and Yang Yang. 2014. Robust hashing with local models for approximate similarity search. *IEEE transactions on cybernetics* 44, 7 (2014), 1225–1236.
[33] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2009. *EMF: Eclipse Modeling Framework 2.0* (2nd ed.). Addison-Wesley Professional.
[34] Martin Steinebach, Peter Klöckner, Nils Reimers, Dominik Wienand, and Patrick Wolf. 2013. *Robust Hash Algorithms for Text*. Springer Berlin Heidelberg, Berlin, Heidelberg, 135–144.
[35] Harald Störrle. 2013. Towards clone detection in UML domain models. *Software & Systems Modeling* 12, 2 (2013), 307–329.
[36] Daniel Struber, Julia Rubin, Gabriele Taentzer, and Marsha Chechik. 2014. Splitting models using information retrieval and model crawling techniques. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 47–62.
[37] Ashwin Swaminathan, Yinian Mao, and Min Wu. 2006. Robust and secure image hashing. *IEEE Transactions on Information Forensics and security* 1, 2 (2006), 215–230.
[38] Khaled Tarmissi and A Ben Hamza. 2009. Information-theoretic hashing of 3D objects using spectral graph theory. *Expert Systems with Applications* 36, 5 (2009), 9409–9414.
[39] Ramarathnam Venkatesan, S-M Koon, Mariusz H Jakubowski, and Pierre Moulin. 2000. Robust image hashing. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, Vol. 3. IEEE, 664–666.