Persistent Naming Based on Graph Transformation Rules to Reevaluate Parametric Specification

Anaïs Cardot[1] , David Marcheix[2] , Xavier Skapin[3] , Agnès Arnould[4] , Hakim Belhaouari[5]

[1]University of Poitiers, XLIM ASALI, anais.cardot@univ-poitiers.fr
[2]ENSMA, LIAS, david.marcheix@ensma.fr
[3]University of Poitiers, XLIM ASALI, xavier.skapin@univ-poitiers.fr
[4]University of Poitiers, XLIM ASALI, agnes.arnould@univ-poitiers.fr
[5]University of Poitiers, XLIM ASALI, hakim.belhaouari@univ-poitiers.fr

Corresponding author: Anaïs Cardot, anais.cardot@univ-poitiers.fr

Abstract. The ability of 3D modeling tools to generate automatically various versions of a similar model is an increasing need in several and different domains such as Archaeology, Architecture or Geology. To address this issue, we propose to use both the *Jerboa* library designed to assist the development of application-specific modelers and the reevaluation methods used for parametric systems in the CAD domain. Unlike most approaches, Jerboa is independent from any application domain and allows rapid development of new operations using graph transformation rules to automatically check the consistency of different objects properties. But it does not support any reevaluation mechanism and especially not a *persistent naming system*, that is used to identify entities of the initial model and match them with entities of the reevaluated model. Using the capacity of graph transformation rules, we address naming problems through a very precise characterization of the basic elements forming the model.

Keywords: Parametric Specification; Persistent Naming; Graph Transformation Rules; Generalized Maps.
DOI: https://doi.org/10.14733/cadaps.2019.985-1002

1 INTRODUCTION

For some years now, the ability of 3D modeling tools to generate automatically various versions of a similar object has been an increasing need in several domains. For example, in the discipline of Archaeology, *Geometry Modeling* based on fragmentary data requires the definition of several restitution hypotheses and the availability of a tool to test these hypotheses quickly and simply.

In CAD domain, *Parametric systems* have been used for many years to enable automatic reevaluation of the construction process, allowing to modify any part of an object construction history and to replay this history

to produce a new result. In the architectural domain, *Procedural generation grammars* is another commonly used process for creating several variants of the same building [11], [20], but it requires some rich corpus information to produce grammars. Moreover, the same tool cannot be used for very different case studies with many specific features.

In this paper, we propose to use the *Graph Transformation Rules* formalism [9] through a *Java* library called *Jerboa* [4], [1], designed to assist the development of application-specific modelers. Rule-based languages form a standard approach for geometric modeling, from plant growth with the seminal L-Systems [19], to numerous applications such as representing the internal structure of wood [17], [21] or buildings [11]. Unlike most approaches, Jerboa is independent from any application domain and avoids any hand-coding of operations, except rule writing. It allows rapid development of new operations to automatically check the consistency of different objects properties. All applications developed with Jerboa share the same topological model called *Generalized maps* (or "*G-maps*") [14], describing a particular class of labeled graphs. Several applications issued from Jerboa and/or G-maps already exist, for different domains such as Architecture [12], Geology [10] and Physics-based modeling [5].

But as it is, Jerboa does not support the rapid production of various similar geometrical models, using the mechanisms of *reevaluation* inherent to parametric systems used in CAD domain. More precisely, a *parametric system* is a two-fold data structure composed of a topologically-based geometric model defining the explicit geometry of the designed object (called *parametric object*), and a mechanism able to reevaluate it when some parameters are changed (called *parametric specification*)[13]. Most current parametric modeling systems are known as "history-based" because the parametric specification may be regarded as an history of modeling functions, which are attached *via* their parameters to topological entities defined in previous states of the model. Such an approach requires ensuring the persistence of the referenced entities and avoiding systems failure during the reevaluation phase when various kinds of topological changes occur. This issue, known as *persistent naming*, is illustrated in Fig. 1. The initial parametric specification is composed of three constructive operations (Fig. 1a): (1) Cube creation; (2) Slot creation on face f (which is split into faces f_1 and f_2); (3) Slot creation on f_2. During reevaluation (Fig. 1b), the first slot is shortened, so f is not split and neither f_1 nor f_2 are created. But since the second slot creation requires f_2 as a parameter, the issue comes down to try and find another entity to replace it. Persistent naming should enable both unambiguous identification of initial model entities and consistent matching between initial and reevaluated model entities (in Fig. 1, f_2 should be matched with f_x).

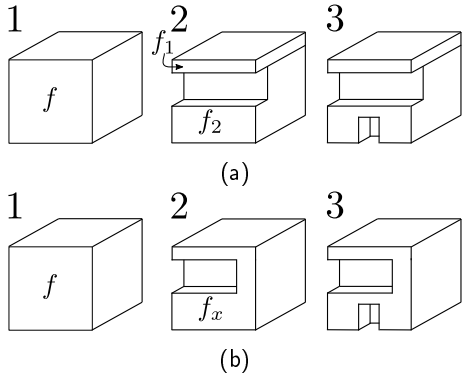

Figure 1: Parametric specification. (a) Initial evaluation. (b) Reevaluation

Persistent naming is a much-debated problem in CAD domain [13], [7], [23], [22], [6], [18], [16], [2],[24], but has never been investigated in conjunction with graph transformation rules. Our approach enables: (1) to extend the persistent naming scope to modeling systems based on such graph transformation rules; (2)

to extend Jerboa by including the working mechanisms of parametric systems. We address naming problems through a very precise characterization of the basic elements forming the model and propose a naming mechanism both general (independent of the model dimension) and homogeneous (independent of the entity dimension), for which only the entities actually used in the parametric specification are followed. Unlike others methods, this follow-up is performed only during reevaluation (and not also during initial evaluation), in order to optimize both time and memory consumption. Moreover, beyond static reevaluation with only parameter modifications, we explore how to carry out *parametric specification edition* (*i.e.* adding, deleting and moving modeling operations).

In Sec. 2, we present the G-maps model, the graph transformation rules and our contribution to persistent naming. In Sec. 3, we detail the different parts of our works, from the persistent naming system to the complete edition of a parametric specification using generic bulletin boards and history records. We conclude in Sec. 4 and propose some perspectives.

2 MAIN CONCEPTS

2.1 Generalized maps

Jerboa is based on the *Generalized Maps* topological model (or *G-Maps* [15]). In a nutshell, G-Maps describe the decomposition of n-dimensional objects according to the successive dimensions of their boundaries, the different parts being linked by relationships noted α_i. More precisely, α_i relationships are *involutions*, that is $\alpha_i \circ \alpha_i = id$. For example, the 2D object in Fig. 2a is split into:

- faces linked by α_2 (blue line, Fig. 2b);
- edges (faces sides) linked by α_1 (red lines, Fig. 2c)
- vertices (ends of edges) linked by α_0 (black lines, Fig. 2d).

A G-Map is therefore a graph whose nodes are called *darts* (represented as green disks in Fig. 2e) and arcs represent various α_i. Entities are described as specific sets of darts linked by dimension-specific α_i: vertices (dim. 0), edges (dim. 1) and faces (dim. 2) are respectively defined as set of darts linked by (α_1, α_2), (α_0, α_2) and (α_0, α_1).

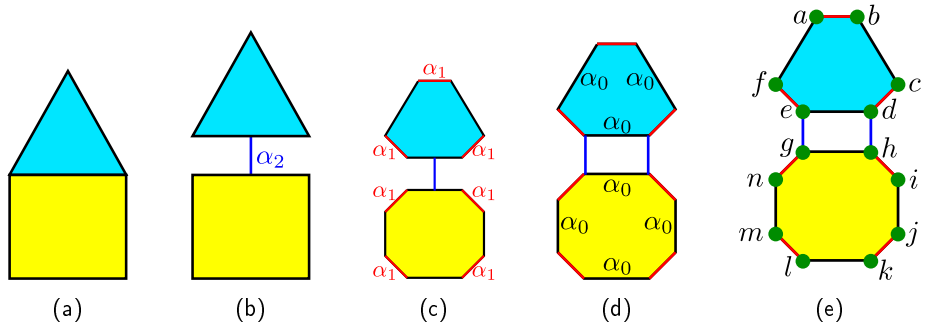

Figure 2: G-Map representation of 2D objects sharing an edge

We call *orbit type* the set $\{\alpha_i, ..., \alpha_n\}$ describing any entity, denoted as $\langle i...n \rangle$: orbit type "Vertex" (resp. "Edge", "Face") shown in Fig. 2 is thus denoted as $\langle 12 \rangle$ (resp. $\langle 02 \rangle$, $\langle 01 \rangle$). We call *orbit* the association of a dart with an orbit type to designate a specific entity. Thus, Fig. 3a to 3c show each Face, Edge and Vertex orbit associated with dart e, denoted respectively $e.\langle 01 \rangle$, $e.\langle 02 \rangle$, $e.\langle 12 \rangle$ and composed, respectively, of 6, 4 and 4 darts. Less "traditional" orbits are defined in the same way; for example, Fig. 3d shows the orbit $e.\langle 1 \rangle$

(the orbit composed of the two darts defining the part of the vertex belonging to the blue face) and the orbit $n.\langle\rangle$ (the orbit composed of the single dart n defining the part of the vertex belonging to the left edge of the yellow face).

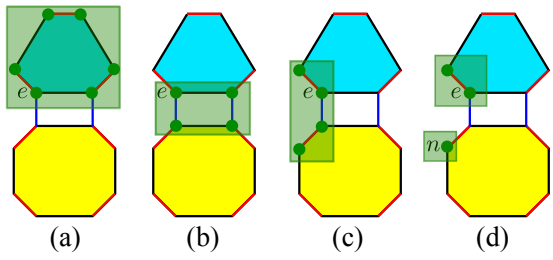

Figure 3: Orbits related to darts. (a) face, (b) edge, (c) vertex and (d) various orbits.

To get a geometrical representation of the object featured in Fig. 2a, some embedding information can be associated with its topological structure. This information is attached to specific orbits. For example, positions are associated with vertices (orbit type $\langle 12\rangle$) to represent geometrical points, while colors are associated with faces (orbit type $\langle 01\rangle$). The ability to define embeddings to less "traditional" orbit types is very useful in many applications. For example,the use of orbit type $\langle 1\rangle$ to model different springs joining on a same vertex, enables a mass-spring system to be easily associated with the geometric model. In our framework, the entities used in the parametric specification will henceforth be expressed as orbits. They can be fully characterized by their type and a selection of their darts. In the rest of the article, we will use an equivalent but more synthetic representation of the G-Maps, as shown in Fig. 3, with α_0 (resp. α_1, α_2) links pictured as black (resp. red, blue) segments and darts symbolized by green disks.

2.2 Graph transformation rules

Jerboa is based on topological rules of graph transformation [4]. Each modeling operation is formally defined as a rule applied on a G-Map. The application of successive rules allows to change the orbits of the G-Map. Jerboa ensures by design that the topological consistency of the G-Map is maintained after each rule application.

Rules are made up of two parts separated by a left-to-right arrow. The left (resp. right) part, which describes the pattern to be filtered (resp. the rewritten pattern), represents the model before (resp. after) application. Patterns are defined by the orbit types of the rule nodes. For instance, Fig. 4a shows the rule describing the creation of a triangular face from scratch: nodes $\mathbf{m_0}$ to $\mathbf{m_5}$ are generated from scratch and are α_0 and α_1 to create a face as the one depicted in Fig. 5a.

Let us suppose we want to apply some triangulation on the face we have just created. We use another rule, namely the Triangulation rule described in Fig. 4b. The left node n_0 carries the orbit type $\langle 01\rangle$, and thus filters the face associated with this node. More precisely: (1) For each dart of the orbit filtered on the left side, as many copies as there are nodes defined on the right side are created: in the case of the Triangulation rule, three copies of each face dart are created. (2) The orbit type associated with each node on the left side is updated for each α_i and for each node on the right side: in Fig. 4b, the orbit type $\langle 01\rangle$ of node $\mathbf{n_0}$ is changed to $\langle 0\_\rangle$ for node $\mathbf{m_0}$: it means that α_0 stays the same, while the initial α_1 is deleted (noted "_"). Similarly, for node $\mathbf{m_1}$ (resp. $\mathbf{m_2}$) on the right side, the initial α_0 is deleted (resp. replaced with α_1); and for both nodes, initial α_1 is replaced with α_2. (3) Finally, nodes on the right side are linked to supply for missing α_i: in Fig. 4b, nodes $\mathbf{m_0}$ and $\mathbf{m_1}$ (resp. $\mathbf{m_1}$ and $\mathbf{m_2}$) are linked by α_1 (resp. α_0).

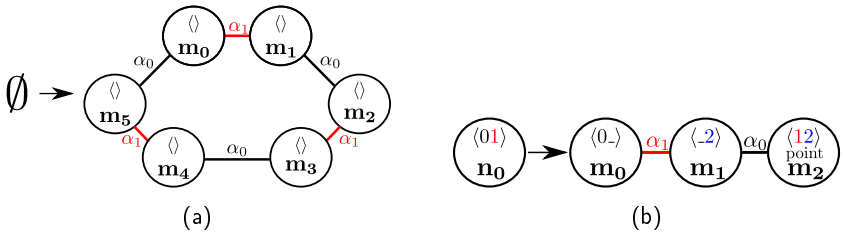

Figure 4: Graph transformation Rules. (a) Triangle creation; (b) Triangulation

Fig. 5 illustrates the creation of a triangular face, followed by the 3-step application of the Triangulation rule on this face. (1) Each dart $a_0, b_0, \ldots f_0$ (Fig. 5a) spawns a triplet of copies $[(a_0, a_1, a_2), (b_0, b_1, b_2), \ldots (f_0, f_1, f_2)]$ (Fig. 5b). Each dart numbered i is the instance of the related node with the same number. (2) The orbit type of each copy is updated (Fig. 5c). (3) The elements of each triplet are linked together by α_0 and α_1 (Fig. 5d).

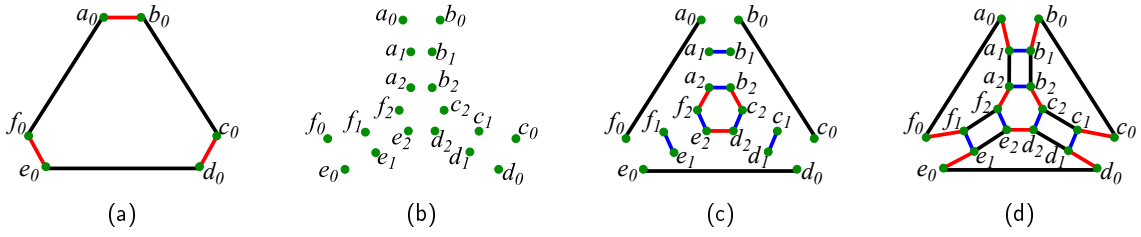

Figure 5: Triangulation rule applied on a face

Moreover, the right side of a rule may contain expressions that define the new embedding of the rewritten pattern. For example, in Fig. 4b, node $\mathbf{m_2}$ contains the `point` expression, that calculates the barycentric center of the filtered face. The operation defined by the rule shown in Fig. 4b is thus the barycentric triangulation.

2.3 Persistent naming

Our method of persistent naming is grounded on both G-Maps and rewriting rules. Persistent naming ([13]) is meant to characterize the topological entities in a reliable way during the initial construction. Parameters of parametric specification operations are often topological references, so this mechanism is essential to produce a valid reevaluation.

Various naming methods have been proposed to try and solve this problem.Most methods ([13], [7], [23], [22], [6], [18], [16], [24]) use faces as reference to name all other entities, since in 3D, each entity can be characterized by an intersection of faces and some additional geometric information [16]. However, these naming algorithms are *not generalizable in dimension* n because this assumption is only true for 3D models. Moreover, the different entities each have their own naming algorithm (for example, a vertex will not be named and matched using the same algorithm as an edge), so *the naming is not truly homogeneous*. It should be noted that the works of Baba-Ali *et al.* [2] are based on edges and are more homogeneous (the same naming and matching algorithm is used for entities whose dimension is higher than 1). In addition, even though the design of persistent naming is well depicted in the literature, the way it can be used for reevaluation is not always precisely defined. Furthermore and despite memory over-consumption, it is usually necessary to trace the evolution of many entities during both initial evaluation and reevaluation, in order to perform the matching between entities when reevaluating, even though many of them will not be used. For example, the naming method proposed by [2], [13] [7], [23], [16] requires to trace or to propagate the evolution of all the

faces and edges of the model. Finally, because persistent naming methods rely on the follow-up of topological entities and thus on the order of the operations applied during initial evaluation, most of published literature focuses on parameters modification, rather than parametric specification edition (adding, deleting or moving an operation before reevaluation); and even in the latter case, few details are provided about inner working. We describe in the next section the various mechanisms that address these limitations.

3 REEVALUATION MECHANISMS

3.1 Parametric specification and edition

To reevaluate a sequence of modeling operations, it is necessary to record them in the form of a parametric specification beforehand. Each operation corresponds to the call of a graph transformation rule as defined in Sec. 2.2. Let us consider the sequence of modeling operations performed in the initial specification, as shown in Fig. 6: under each operation, the result of the operation is displayed in both topological (top) and geometric (bottom) representation. In a classic way, operation parameters correspond to darts defined at previous steps. But, the specification cannot be limited to the simple recording of rule calls (physical identifiers of darts being inherently unstable from one reevaluation to another, they cannot be used directly). Darts must therefore be **labeled persistently**. The use of rules makes it possible, both in the initial evaluation and the reevaluation, to assign each dart a *Persistent Id*, rated PI_a, PI_b, ... for darts a, b and so on.

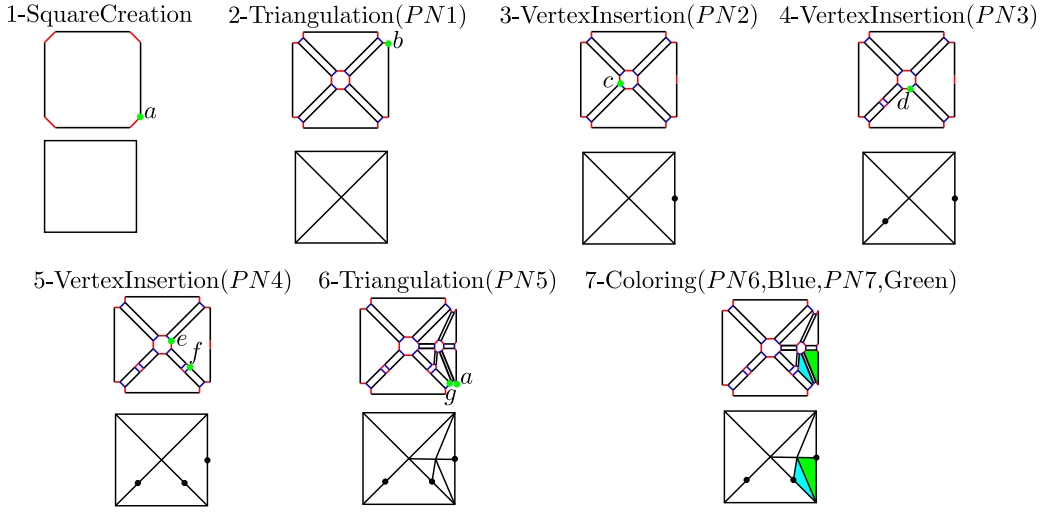

Figure 6: Initial specification

Rules are defined for any filtered orbit, but only specific orbits are used by modeling operations as parameter entities. To identify each of these entities in a persistent way, we define the *Persistent Names (PN)* of entities. Each PN is composed of a set of Persistent Identifiers to keep track of all operations that have impacted that entity (see Sec. 3.2.2). More precisely, $PN = \{PI\}.\langle o \rangle$, where $\{PI\}$ is a set of Persistent Ids of the representative darts of the orbit, and $\langle o \rangle$ is the orbit type of the entity. For example, $\{PI_a\}.\langle 01 \rangle$ is the Persistent Name of the face orbit $a.\langle 01 \rangle$, where a is a dart.

The parametric specification shown in Fig. 6 is: 1-SquareCreation; 2-Triangulation($PN1$); 3-VertexInsertion($PN2$); 4-VertexInsertion($PN3$); 5-VertexInsertion($PN4$); 6-Triangulation($PN5$); 7-VertexInsertion($PN6$, Blue, $PN7$, Green), where $PN1$, ..., $PN7$ are respectively the Persistent Names containing the Persistent Ids detailed in Table 1. Note that $PN5$, which represents the face triangulated by

the sixth operation, contains at the same time PI_e, PI_b and PI_f: indeed, darts e, b and f all belong to the same face and keep track of 1-SquareCreation and 2-Triangulation; but b also registers 3-VertexInsertion while f registers 5-VertexInsertion. e doesn't register another operation, but it is kept as the dart designated by the user to select the face. These PI are therefore complementary to characterize the rules that have impacted the face and must be kept together.

PN	PI	O. type	PN	PI	O. type
$PN1$	$\{PI_a\}$	$\langle 01\rangle$	$PN5$	$\{PI_e, PI_b, PI_f\}$	$\langle 01\rangle$
$PN2$	$\{PI_b\}$	$\langle 02\rangle$	$PN6$	$\{PI_g\}$	$\langle 01\rangle$
$PN3$	$\{PI_c\}$	$\langle 02\rangle$	$PN7$	$\{PI_a\}$	$\langle 01\rangle$
$PN4$	$\{PI_d\}$	$\langle 02\rangle$			

Table 1: Persistent Ids and Orbit types related to operation parameters of the initial specification.

To illustrate the behavior of our persistent naming mechanism in case of some parametric specification edition, we modify the initial specification by placing 2-Triangulation after 4-VertexInsertion. This illustrates our method to add, delete and move operations, as adding and deleting operations are encompassed by moving operation. The reevaluation proceeds as shown in Fig. 7. For each operation, topological parameters PN_i need to be matched to darts in reevaluated model, in order to call the corresponding rule. Sec. 3.2.5 will describe in a precise way how such a matching is done.

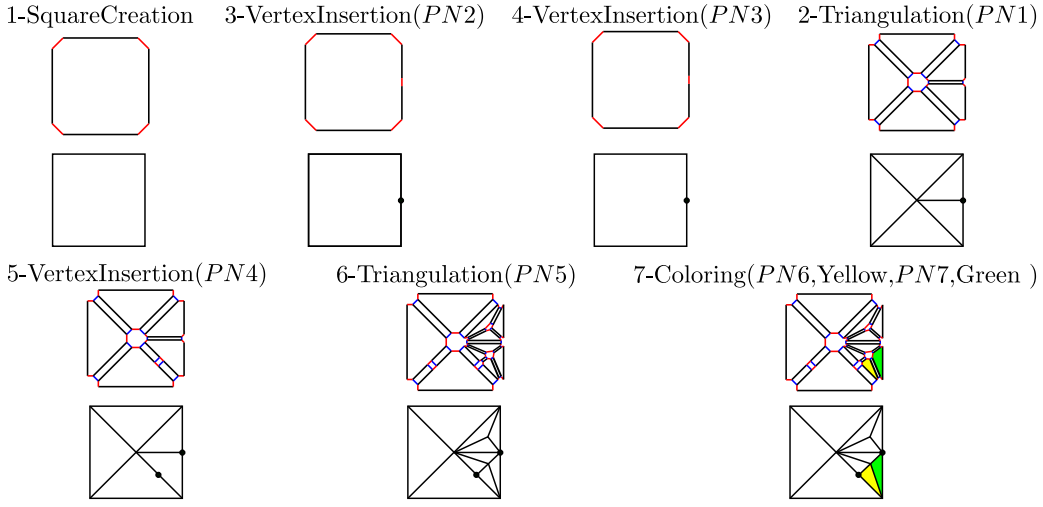

Figure 7: Specification reevaluation

1. 1-SquareCreation is reevaluated the same way as in the initial evaluation (the related rule is applied).
2. The parameter $PN2$ of 3-VertexInsertion (corresponding to $\{PI_b\}$ in the initial set) needs to be matched to darts in the reevaluated model. For this operation, 2-Triangulation is considered as deleted, as it is moved after it.
3. 4-VertexInsertion is reevaluated. Just as 3-VertexInsertion, it considers 2-Triangulation as deleted. Matching $PN3$, we find a dart representing the edge and call the related rule.

4. 2-Triangulation has been moved. This operation considers 3-VertexInsertion and 3-VertexInsertion as added operations, as they are now applied before. Using $PN1$, the matching process finds a dart representing the face and applies the related rule.

5. 5-VertexInsertion is also reevaluated. Using $PN4$, we find a dart representing the edge and call the related rule.

6. 6-Triangulation is reevaluated. We use $PN5$ to find the new darts representing the face.

7. Finally, 7-Coloring is reevaluated with a change to one of its color parameters.

Inside the parametric specification, it is possible to characterize the effect of edition process on blocks of modeling operations, with respect to their relative position to the edited operation. Fig. 8 shows these different blocks according to the type of edition (adding, deleting, moving up or moving down of a modeling operation *OP*). For example, as shown in Fig. 8c, moving down *OP* from position *p* to *q* subdivides the parametric specification in 3 blocks. For every operation **before** *p*, this moving has no effect upon the reevaluation process. For every operation **between** *p* and *q*, *OP* can be seen as "deleted" during the reevaluation process (*OP* being reevaluated later at position *q*). For moved operation *OP* itself, all intermediary operations can be considered as "added". Finally, every operation located **after** the new position *q* considers *OP* as "moved".

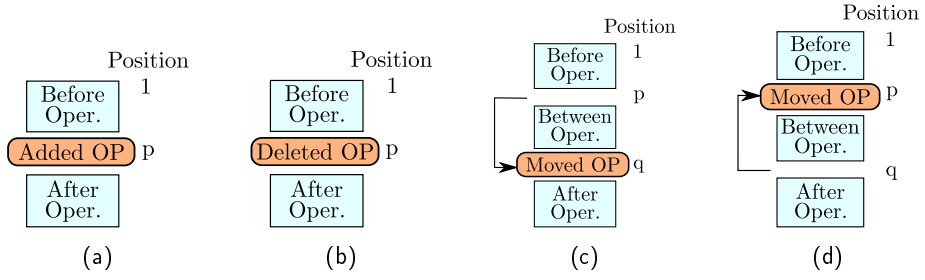

Figure 8: Different types of edition in a parametric specification. (a) adding, (b) deleting, (c) moving down, and (d) moving up an operation.

As shown above, determining the types of edition undergone by operations is mandatory to apply the reevaluation. But achieving entities matching also requires to determine how the Persistent Names of referenced orbits have evolved.

3.2 Orbit evolution

We consider the evolution of orbits for both initial evaluation and reevaluation. First, we distinguish the different types of orbital evolution that may happen (Sec. 3.2.1). Then, in order to match evaluation and reevaluation entities, we detail the structures of related Persistent Ids and Persistent Names (Sec. 3.2.2). Finally, we propose a structure allowing to follow the entities during the evaluation and a tree structure allowing to report the matching during the reevaluation (Sec. 3.2.3 to 3.2.5).

3.2.1 Evolution types

We define the following types of orbit evolutions, some of which are shown in Fig. 6 and 7. (a) *Creation*: creates a new orbit. (b) *Deletion*: removes an orbit, so no modeling operation can use it anymore. (c) *Fusion*: merges several orbits. (d) *Modification*: modifies the orbit without any splitting or merging. (e) *NoEffect*: does not affect the orbit. (f) *Split*: splits the orbit.

3.2.2 Persistent naming

The Persistent Id (PI) of a dart is set at the time of dart creation, and then modified each time the dart is rewritten by rules. Each PI consists of the various operation numbers and rule nodes that have created or rewritten the related dart. For instance, consider the operation 6-Triangulation in the initial specification. This operation takes $PN5$ as a parameter, and $PN5$ refers to PI_c, PI_b and PI_f, according to Table 1. If we focus on PI_b for example, we note that before applying 6-Triangulation, b has been modified by the node $\mathbf{m_0}$ of the rule defining 3-VertexInsertion (Fig. 9): "$3-\mathbf{m_0}$" is thus a part of PI_b.

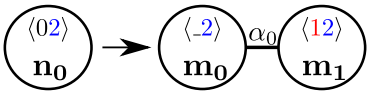

Figure 9: Vertex Insertion rule

Before that, the face b belongs to has been triangulated when applying 2-Triangulation: the dart had been modified by the node $\mathbf{m_0}$ of the related rule, leading to another part of PI_b : "$2-\mathbf{m_0}$". Finally, the rule corresponding to the first operation, 1-SquareCreation, has created the dart *via* one of its nodes; the rule for Square creation is quite similar to the one related to Triangle creation (Fig. 4a): to create a square, 8 nodes (denoted as $\mathbf{m_0}$, ... $\mathbf{m_7}$) are created from scratch and linked by a succession of α_0 and α_1. In the current case, node $\mathbf{m_3}$ corresponds to the dart we are interested in: the related part of PI_b is "$1-\mathbf{m_3}$". Thus, concatenating those operations in chronological order, PI_b is defined as $\{1-\mathbf{m_3}; 2-\mathbf{m_0}; 3-\mathbf{m_0}\}$ before 6-Triangulation is applied. The same procedure is applied to define PI_a, ... PI_f.

Note that before applying 3-VertexInsertion, PI_b was defined as $\{1-\mathbf{m_3}; 2-\mathbf{m_0}\}$, so $PN2$ and $PN5$ do not have the same definition as PI_b. The same goes for PI_a when used in $PN1$, which is defined differently from PI_a when used by $PN7$.

Dart b has been in the model since its creation, but darts which have been created later also have a *PI*. Let us take the example of dart c, created by 2-Triangulation: c has been created by instantiating the node $\mathbf{m_2}$ of the rule defining 2-Triangulation, but $\mathbf{m_2}$ itself results from the rewriting of node $\mathbf{n_0}$ (Fig. 4b), which is associated with a dart defined as $\{1-\mathbf{m_7}\}$. Therefore, PI_c is defined as $\{1-\mathbf{m_7}; 2-\mathbf{m_2}\}$.

PN are used as operation parameters (see Sec. 3.1). Thus, 3-VertexInsertion, which inserts a vertex in the edge adjacent to dart b, has face's Persistent Name $PN2 = \{\{1-\mathbf{m_3}; 2-\mathbf{m_0}\}\}.\langle 02\rangle$ as topological parameter. 6-Triangulation is also applied to an orbit adjacent to b, a face. However, $PN5$ is different from $PN2$ because the face (and therefore b) has been affected by 3-VertexInsertion as seen previously. What's more, as seen in Sec. 3.1, e and f must also be used to describe this face (the first one because it was designated by the user during initial construction to apply the triangulation, the second one because it is modified by 5-VertexInsertion, which doesn't modify e or b). Thus, $PN5 = \{\{1-\mathbf{m_3}; 2-\mathbf{m_2}\}, \{1-\mathbf{m_3}; 2-\mathbf{m_0}; 3-\mathbf{m_0}\}, \{1-\mathbf{m_4}; 2-\mathbf{m_1}; 5-\mathbf{m_1}\}\}.\langle 01\rangle$.

Note that this naming mechanism is effective in an homogeneous way for all orbits, whatever the topological dimension of the modeled objects. Indeed, identical persistent names are used for all entities dimension and defined in a general way regardless of the model dimension.

3.2.3 Generic rule bulletin boards

Following orbit evolutions over the specification entails determining how each operation partakes in this evolution. We use structures called *generic bulletin boards* for that purpose.

Some commercial modelers like 3D ACIS provide bulletin boards to track entity creation, alteration, or deletion ([8]). Indeed, high-level operations provided by APIs (boolean operations, chamfering, and so on) can generate important modifications on topological entities. Bulletin boards, generated by those modelers' kernel,

can be read after each operation and the way to use them is well described in the APIs' documentation. But, because it's proprietary information and despite the fact that it's essential to any monitoring system, little is known on how exactly bulletin boards are generated.

Baba-Ali *et al.* ([3], [2]) uses G-Maps to build a bulletin board by associating each dart with two 4-tuples of integers. The first (resp. second) 4-tuple contains the number of topological orbits the dart belongs to before (resp. after) applying an operation. By comparing the pair of integers related to the same orbit, it can be determined whether the corresponding entity has been modified or not and the event is recorded into the bulletin board. This approach is generic in all dimensions but requires to go through the sets of integers before and after each modeling operation.

Our approach is rule-specific: a generic bulletin board is generated when the user creates a rule to account for the different types of evolution (Sec. 3.2.1). Fig. 10 shows the generic bulletin board for the Triangulation operation. There is one box per orbit type. Inside each box, we describe the evolution types for the rewritten nodes. Let $\langle x \rangle$ be an orbit type ($\langle\rangle$, $\langle 1 \rangle$ and so on): we gather the nodes of the right side of the rule, whose rewriting instantiates darts belonging to the same $\langle x \rangle$, then we search for the left-side nodes which have rewritten these darts, and for which orbit type. A tree is then created for each set: the root contains the nodes selected on the right side, and the leaves contain left-side nodes and the related orbit. The joining arcs are labeled with the type of evolution carried out.

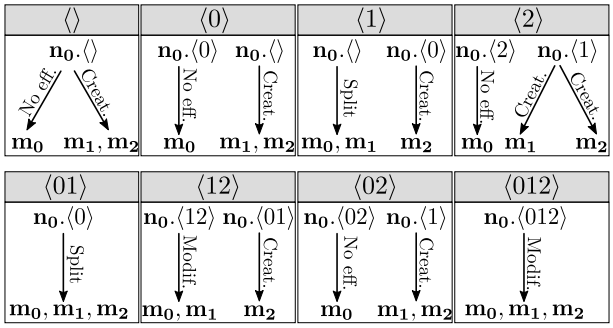

Figure 10: Generic Triangulation bulletin board

As an example, consider the orbit type $\langle 12 \rangle$ in the generic bulletin board displayed in Fig. 10. Fig. 5 shows the output of the Triangulation rule applied on a triangular face. Fig. 11 focuses on vertices (orbit type $\langle 12 \rangle$): three vertices are composed of darts instantiated by nodes $\mathbf{m_0}$ and $\mathbf{m_1}$ whereas the central vertex is made of darts instantiated by node $\mathbf{m_2}$. Each vertex of the pair ($\mathbf{m_0}$, $\mathbf{m_1}$) is created with a pair of darts instantiated by $\mathbf{n_0}$ on the left side of the rule and linked by $\langle 12 \rangle$. The type of evolution of these vertices is a modification, because we simply add darts to an already existing vertex. A tree is thus created with root labeled "$\mathbf{n_0}.\langle 12 \rangle$" and leaf labeled "$\mathbf{m_0}$, $\mathbf{m_1}$", linked by the "Modification" arc. The other darts (forming the central vertex) are issued from a dart instantiated by $\mathbf{n_0}$ on the left side of the rule and are bound by $\langle 01 \rangle$. A second tree is thus created, with root labeled "$\mathbf{n_0}.\langle 01 \rangle$" and leaf labeled "$\mathbf{m_2}$", linked by the "Creation" arc.

3.2.4 History Record

Generic bulletin boards are completed by *history records* to process the whole specification. History records analyze the successive generic bulletin boards of the rules that have impacted any dart. One carries out as many history records as there are PI. Let $PN = \{PI_a, PI_b, \dots\}.\langle x \rangle$ be a Persistent Name. Let $PI_b = \{1 - \mathbf{m_i}; ...; k - \mathbf{m_j}\}$ be the Persistent Id of dart b. To create the history record of PI_b, we scan its contents in reverse order (from the most recent to the oldest). Therefore, we first consider $\langle x \rangle$ and $k - \mathbf{m_j}$ (the last

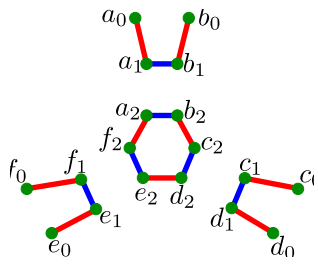

Figure 11: Topological view of face vertices

rewriting of dart b by the node $\mathbf{m_j}$ of the related rule set at step k). In the generic bulletin board of this rule, we retrieve the box corresponding to $\langle x \rangle$ and we select the (unique) tree whose child contains $\mathbf{m_j}$. This process is then repeated by going back up each operation constituting PI_b, knowing that it retrieves, for the $(k-1)$-th operation, the box of the generic bulletin board corresponding to the orbit indicated at the root of the tree used for operation k.

To illustrate this point, let us create the history record for 6-Triangulation applied to $PN5$ (see Fig. 6 and 7), that has $\{PI_b\}$ as one of its Persistent Ids (see Table 1). The result is shown in Fig. 12, with green and red arrows labeling the 6-step process. As seen in Sec. 3.2.2, $PI_b = \{1-\mathbf{m_3}; 2-\mathbf{m_0}; 3-\mathbf{m_0}\}$ before applying 6-Triangulation. $PN5 = \{PI_e, PI_b, PI_f\}.\langle 01 \rangle$, meaning that 6-Triangulation is to be applied to orbit $\langle 01 \rangle$ (see the bottom of Fig.12a). Assume the last element of PI_b (*i.e.* $3-\mathbf{m_0}$, which is 3-VertexInsertion applied through $\mathbf{m_0}$) has been initially recovered. Step 1: we look at orbit type $\langle 01 \rangle$ in the generic VertexInsertion's bulletin board, that is the last rule having impacted b before triangulating. At this stage, b is rewritten by node $\mathbf{m_0}$. Step 2: The excerpt of the generic bulletin board of VertexInsertion in Fig. 12a shows that, for orbit type $\langle 01 \rangle$, $\mathbf{m_0}$ results from a modification of $\mathbf{n_0}.\langle 01 \rangle$. Step 3: using this orbit type $\langle 01 \rangle$ as an index in

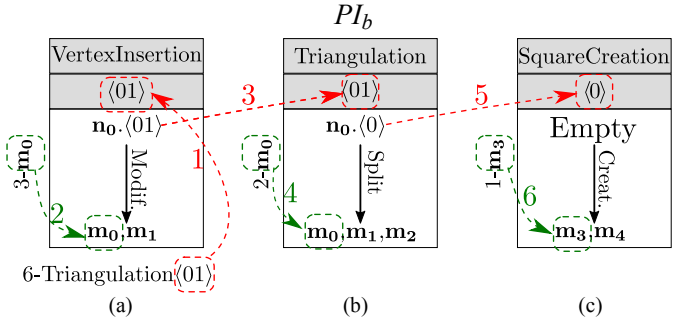

Figure 12: History record of $PN5$ for $PI_b = \{1-\mathbf{m_3}; 2-\mathbf{m_0}; 3-\mathbf{m_0}\}$

the generic bulletin board of the previous operation recorded (that is, 2-Triangulation), we search among the trees related to this entry, the one which contains $\mathbf{m_0}$, since the corresponding identifier in PI_b is $2-\mathbf{m_0}$ (Step 4). We obtain a tree with $\mathbf{n_0}.\langle 0 \rangle$ as root (Fig. 12b). We repeat the process once again: at Step 5, we go through the generic bulletin board associated with the previous recorded operation (1-SquareCreation). Using the orbit type $\langle 0 \rangle$ as an entry, we search for the related tree which contains $\mathbf{m_3}$, since the corresponding identifier is $1-\mathbf{m_3}$ (Step 6). This tree has *Empty* as root (see Fig. 12c), meaning that there is no previous operation. To get the complete history record of $PN5$, we add the history records of both PI_e and PI_f.

Every Persistent Name's history record is carried out in a similar way. Fig. 13 shows the history records related to some of the Persistent Names we use.

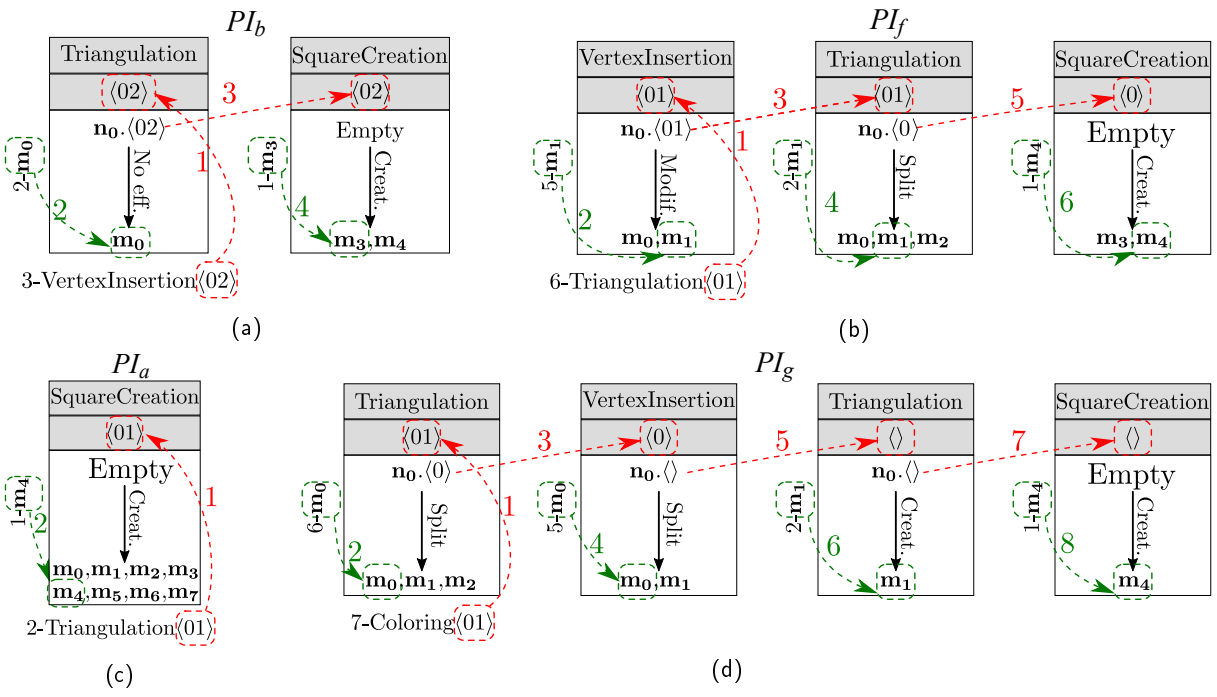

Figure 13: History records of (a) $PN2$ for $PI_b = \{1 - \mathbf{m_3}; 2 - \mathbf{m_0}\}$, (b) $PN5$ for $PI_f = \{1 - \mathbf{m_4}; 2 - \mathbf{m_1}; 5 - \mathbf{m_1}\}$, (c) $PN1$ for $PI_a = \{1 - m_4\}$, (d) $PN6$ for $PI_g = \{1 - \mathbf{m_4}; 2 - \mathbf{m_1}; 5 - \mathbf{m_0}; 6 - \mathbf{m_0}\}$.

3.2.5 Entity matching

Performing reevaluation entails matching evaluation entities with reevaluation entities and we refer, in this section, to operations shown in Fig. 7 to describe various scenarios. In order to perform this matching, for each history record, a *matching tree* is built, with a Persistent Id as root and orbits as leaves. A matching tree allows to determine which darts of the reevaluation will be used for each orbit designated in the initial specification.

For each modeling operation, branches of the matching trees impacted by the related rule are updated by using the orbits stored in the leaves and parts of the corresponding history records. When a specific edition occurs, the way to construct the matching trees depends on the type of edition which has impacted each operation (see Fig. 8).

In case of deletion (for example, operation 2 is considered as deleted by operations 3 and 4 after it has been moved down, or operations directly deleted by the user during reevaluation), the impacted tree branches are updated with a label prefixed by *"not"* in order to indicate that the corresponding parts of history records have not been applied.

In case of addition (for example, operations 3 and 4 are considered as deleted by operation 2 after it has moved down, or operations directly added by the user during reevaluation), the generic bulletin board of the related rule is used to update the matching trees according to the orbits impacted by this addition.

In case of moving (for example, operation is 2 considered as having moved by operations 5, 6 and 7), the history records of all the operations impacted by and after the moved operation are rebuilt for the intermediary operations and the moved one, by taking into account their new positions.

To illustrate this different cases, we now detail step by step the reevaluation in Fig. 7 for every persistent

name.

1-**SquareCreation reevaluation**

Since this operation has no parameter, it is reevaluated in the same way as for the initial evaluation. As the matching trees of $PN1$ to $PN7$ are all impacted by this operation, they are all updated. Fig. 14 shows the model after applying the rule, and the impact on the matching trees. History records shown in Fig. 13 are scanned, one operation after another, to match darts and orbits in the reevaluated model. Consider $PN2$ for instance (see Fig. 14): the history record of PI_b (Fig. 13a) indicates that to process 1-SquareCreation, one must find the newly created orbit type $\langle 02 \rangle$, associated with the instance of node $\mathbf{m_3}$. A branch of the matching tree labeled *"Creat."* (for *"Creation"*) is thus built, related to the orbit found in the reevaluated model ($b'.\langle 02 \rangle$ is indeed the orbit corresponding to the single dart b' which instantiates $\mathbf{m_3}$). Similarly for all other Persistent Names, one matching tree is generated for each dart's Persistent Id.

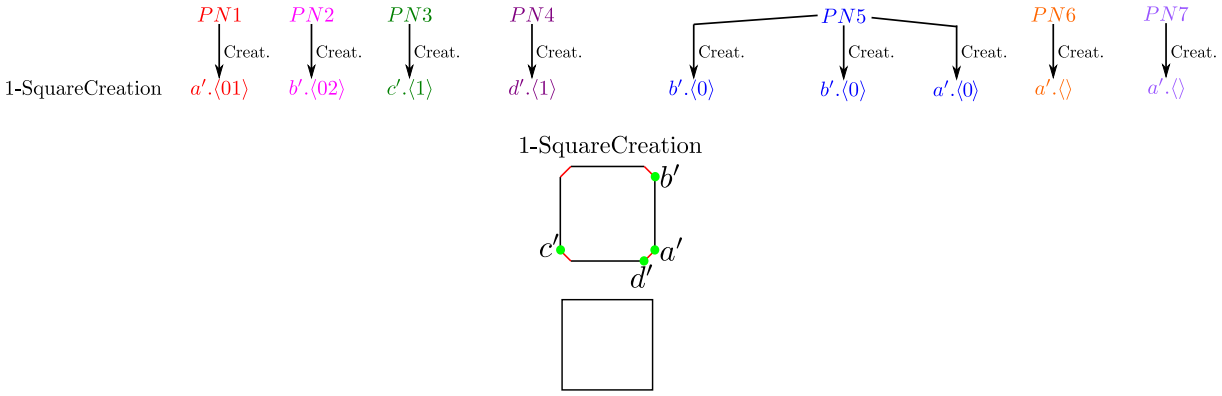

Figure 14: Matching trees (left), topological and geometric model (right) after 1-SquareCreation reevaluation

2-**Triangulation reevaluation**

This operation is moved. For 3-VertexInsertion and 4-VertexInsertion, 2-Triangulation is considered as deleted. Fig. 16 shows that the trees corresponding to $PN2$ and $PN3$ (the Persistent Names used by operations 3 and 4) have been updated (the prefix *"not"* means that the operation has not been applied).

As said previously, for operations 5, 6 and 7 positioned after the moved operation (2) and if the Persistent Names (*PN4, PN5, PN6 and PN7*) contain this operation (*i.e.* the number 2 appears in the *PI* associated with the Persistent Names), we modify the Persistent Names according to the new order of the operations and we recalculate the corresponding history record. For example, the moved operation 2 appears in $PN7 = \{\{1-\mathbf{m_4}; \boxed{2}-\mathbf{m_0}; 3-\mathbf{m_0}; 6-\mathbf{m_0}\}\}.\langle 01 \rangle$. The modified name is $PN7 = \{\{1-\mathbf{m_4}; 3-\mathbf{m_0}; \boxed{2}-\mathbf{m_0}; 6-\mathbf{m_0}\}\}.\langle 01 \rangle$. We recreate the corresponding history record as we have created the original one (see Sec. 3.2.4). The result for $PN7$ is illustrated in Fig. 15. The branch label *"Change"* indicates that the following sub-tree corresponds to a recreated history record. We do the same for $PN5$ and $PN6$.

After the history records have been changed, we select every dart of the former orbit to keep following every possible outcome. For example, the orbit $b'.\langle 0 \rangle$ in $PN5$ contains both darts a' and b', so when we change the history record, we select a' and b' as leafs for the considered tree. As can be seen in Fig. 16, the label "Change" on several arcs means that the related history record has been modified.

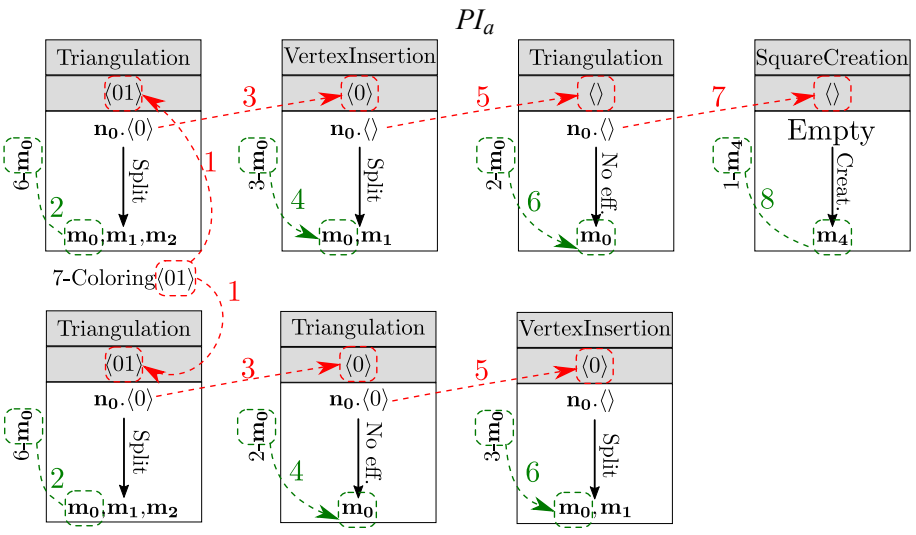

Figure 15: Recalculating $PN7$'s history record. Top: History record built using the initial Persistent Name. Bottom: update according to the new positions of operations

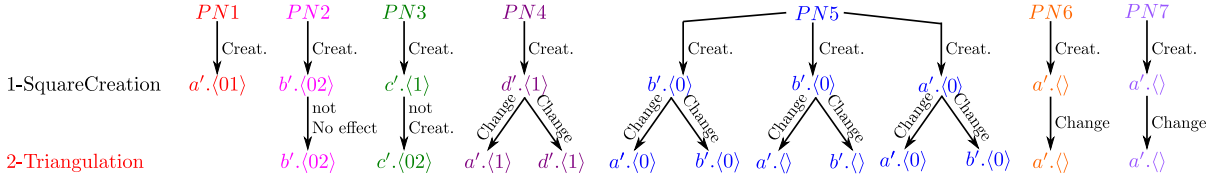

Figure 16: Matching trees (left), topological and geometric model (right) after moving 2-Triangulation.

3-VertexInsertion reevaluation

This operation is reevaluated on $PN2$. Since $PN2$ is no longer used afterwards, its matching tree can now be removed. For 2-Triangulation has *PN1* as topological parameter, the operation is considered as an addition. At this step (*i.e.* just after applying 1-SquareCreation), the leaf of $PN1$ is labeled $a'.\langle 01\rangle$, so we look at the generic bulletin board of Vertex Insertion (Fig. 17) for the type of orbit $\langle 01\rangle$, in order to update the matching tree. The instance we take is chosen among the resulting nodes (here, $\mathbf{m_0}$ and $\mathbf{m_1}$): we decide to keep a', which is an instance of $\mathbf{m_0}$.

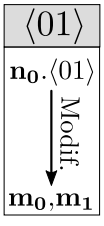

Figure 17: Generic bulletin board excerpt of the VertexInsertion rule

Matching trees are updated accordingly, as shown in Fig. 18: a branch has been added to the tree related to $PN1$. This branch is labeled "add Modif.", meaning that the vertex insertion has modified the orbit type

$\langle 01 \rangle$ (that is, the face) associated with a'.

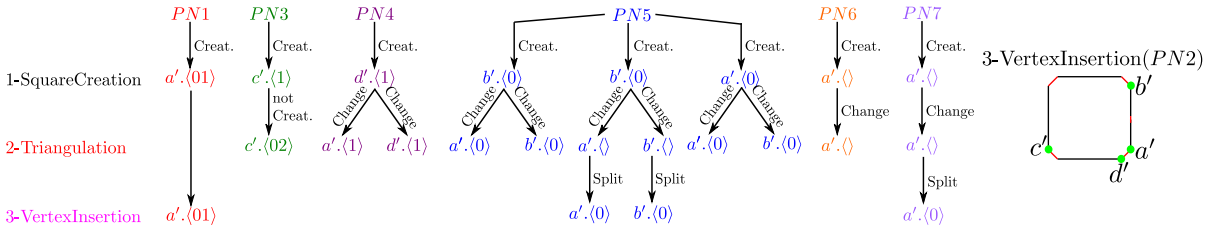

Figure 18: Matching trees (left), topological and geometric model (right) after 3-VertexInsertion reevaluation

4-VertexInsertion reevaluation

This operation should be reevaluated on $PN3$ only. But $PN3$ is matched through a creation which has not been applied. In such a case, we consider that the tree is not to be followed. Therefore, we do not apply 4-VertexInsertion and the matching tree of $PN3$ is not processed anymore. The result of this reevaluation is displayed in Fig. 19.

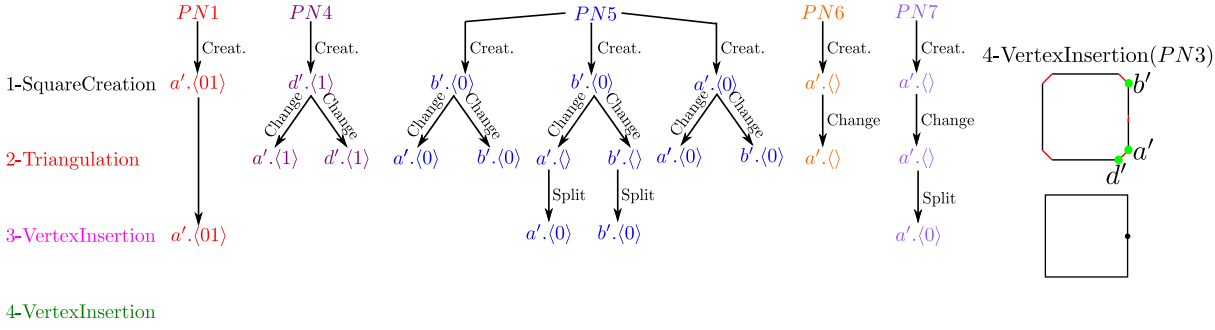

Figure 19: Matching trees (left), topological and geometric model (right) after 4-VertexInsertion reevaluation

2-Triangulation reevaluation

This operation is reevaluated as would be any other, the fact that it is moved has an impact only when the decision to move it has been taken (3 steps ago in this case). The result is shown in Fig. 20.

5-VertexInsertion and 6-Triangulation reevaluation

Those operations are reevaluated. We follow the history records of the Persistent Names left to get the new darts and orbits that match them. The result is shown in Fig. 21.

7-Coloring reevaluation

This operation colors blue the face designated by $PN6$, and colors green the one designated by $PN7$. Here, the parameter "Blue" is changed to "Yellow". Since it is the last operation of the specification, no more matching tree is to be updated. The result is illustrated in Fig. 22.

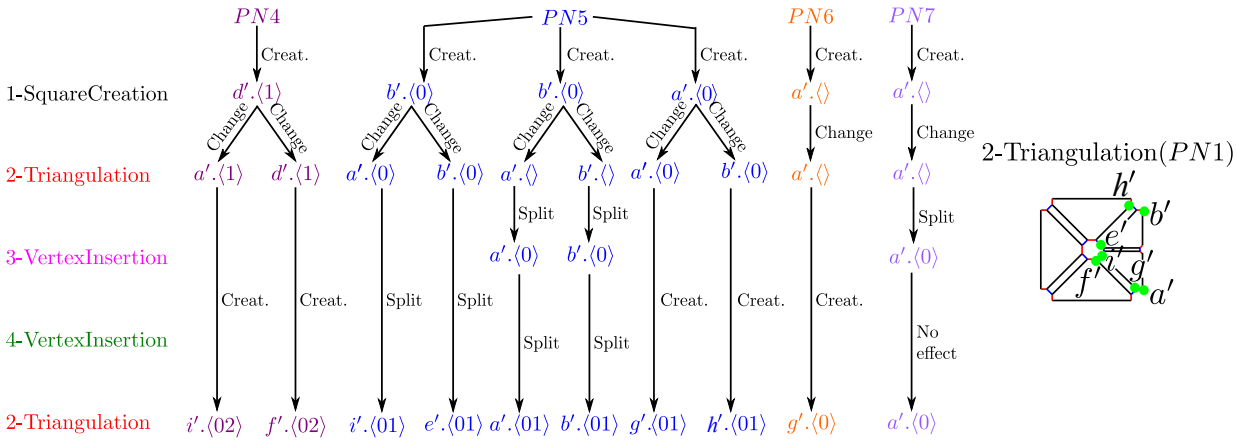

Figure 20: Matching trees (left), topological and geometric model (right) after 2-Triangulation reevaluation.

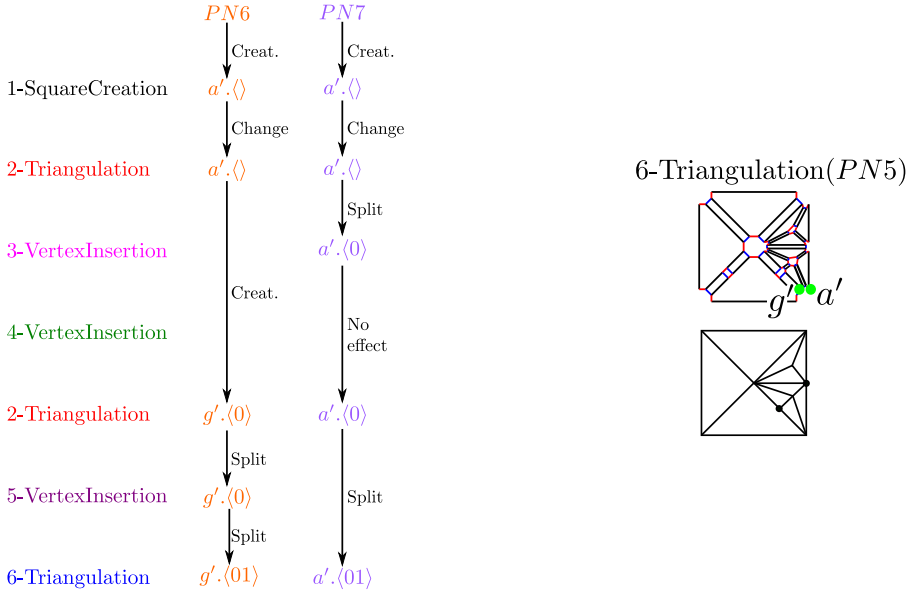

Figure 21: Matching trees (left), topological and geometric model (right) after 5-VertexInsertion and 6-Triangulation reevaluation

4 CONCLUSION AND PERSPECTIVES

In this paper, we present a new persistent naming system and an entity matching algorithm combining the strong points of graph transformation rules and G-Maps to create and reevaluate new models using parametric specification. Using those tools, we lay the foundation to develop 3D modeling systems independent from any application domain and avoid any hand-coding of operations, except rules writing.

The proposed persistent naming system is based on Jerboa and graph transformation rules. But currently, Jerboa only provides the use of elementary rules, such as object creation or deletion, split of topological entities, triangulation, extrusion, and so on. Jerboa is still under development to allow the manipulation of

7-Coloring($PN6$,Yellow,$PN7$,Green)

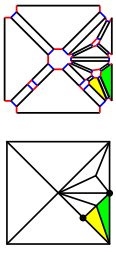

Figure 22: Topological and geometric model after 7-Coloring reevaluation

more complex rules, based on rules scripts that combine many elementary rules. This update will enable a real and precise comparative study with other methods in terms of computation times and storage needs, that is difficult to evaluate at the moment with only elementary rules. However, even if it is only used on elementary rules, our article lays the foundations of the naming system, and shows the mechanisms used and the advantages of this approach. It makes it possible to extend the persistent naming scope to modeling systems based on such graph transformation rules and to extend Jerboa by including the working mechanisms of parametric systems. Through a very precise characterization of the basic elements forming the model, it enables naming all types of entities in an homogeneous (independent of the entity dimension) and general (whatever the dimension of the model) way. It defines unambiguously *Persistent Identifiers* of darts and *Persistent Names* of entities, using the information returned by transformation rules. Unlike most others methods, it follows the evolution of a limited number of entities during reevaluation, in order to achieve matching. Only entities that are actually referenced in the parametric specification are traced, and only during the reevaluation phase. This allows us to hope for space and time savings, but as stated before, a comparative study will have to be carried out in future works. To follow entity evolution after applying an operation, we defined precisely the mechanisms used to track each orbit, that consist in using the generic bulletin board associated with the rule defining this operation. At this time, generic rule bulletin boards have to be designed by the (human) rule designer; it would be interesting to generate them automatically. Finally, beyond static reevaluation with only parameter modifications, we show the impact of parametric specification edition (*i.e.* adding, deleting and moving operations) on the persistent naming system, and we propose in this paper an accurate method to carry out this edition.

ORCID

David Marcheix, http://orcid.org/0000-0003-4062-4232
Hakim Belhaouari, http://orcid.org/0000-0003-4454-7756

REFERENCES

[1] Jerboa. http://xlim-sic.labo.univ-poitiers.fr/jerboa/.

[2] Baba-Ali, M.: Système de nomination hiérarchique pour les systèmes paramétriques. Ph.D. thesis, 2010.

[3] Baba-Ali, M.; Marcheix, D.; Skapin, X.; Bertrand, Y.: Generic computation of bulletin boards into geometric kernels. Conference on Virtual Reality, Computer Graphics, Visualization and Interaction in Africa, 85–93, 2007.

[4] Belhaouari, H.; Arnould, A.; Le Gall, P.; Bellet, T.: A graph transformation library for topology-based geometric modeling. Conference on Graph Transformation, 269–284, 2014.

[5] Ben Salah, F.; Belhaouari, H.; Arnould, A.; Meseure, P.: A general physical-topological framework using rule-based language for physical simulation. 12th International Conference on Computer Graphics Theory and Applications, 2017. http://doi.org/10.5220/0006119802200227.

[6] Bidarra, R.; Nyirenda, P.; Bronsvoort, W.: A feature-based solution to the persistent naming problem. Computer Aided Design and Applications, 2(4), 517–526, 2005.

[7] Capoyelas, V.; Chen, X.; Hoffmann, C.: Generic naming in generative, constraint-based design. Computer-Aided Design, 28(1), 17–26, 1996.

[8] Corney, J.; Lim, T.: 3D Modeling with ACIS. Paul & Company Pub Consortium, 2001. ISBN 9781874672142. https://books.google.fr/books?id=x_JMSgAACAAJ.

[9] Ehrig, H.; Ehrig, K.; Prange, U.; Taentzer, G.: Fundamentals of Algebraic Graph Transformation, 2006.

[10] Gauthier, V.; Arnould, A.; Belhaouari, H.; Horna, S.; Perrin, M.; Poudret, M.; Rainaud, J.F.: A topological approach for automated unstructured meshing of complex reservoir. 15th edition of the European Conference on the Mathematics of Oil Recovery, 2016. http://doi.org/10.3997/2214-4609.201601789.

[11] Haegler, S.; Muller, P.; Van Gool, L.: Procedural modeling for digital cultural heritage. Journal on Image and Video Processing - Special issue on image and video processing for cultural heritage, 2009.

[12] Horna, S.; Meneveaux, D.; G., D.; Bertrand, Y.: Consistency constraints and 3d building reconstruction. Application of Graph Transformations with Industrial Relevance, 41(1), 13–27, 2009. http://doi.org/10.1016/j.cad.2008.11.006.

[13] Kripac, J.: A mechanism for persistently naming topological entities in history based parametric solid models. Proceedings of the 3rd ACM symposium on Solid Modeling and Applications, 21–30, 1995. http://doi.org/10.1145/218013.218024.

[14] Lienhardt, P.: Topological models for boundary representation: a comparison with n-dimensional generalized maps. Computer-Aided Design, 23(1), 59–82, 1991. http://doi.org/10.1016/0010-4485(91)90082-8.

[15] Lienhardt, P.: N-dimensional generalized combinatorial maps and cellular quasi-manifolds. International Journal of Computational Geometry and Applications (IJCGA), 04, 275–324, 1994.

[16] Marcheix, D.: A persistent naming of shells. International Journal of CAD/CAM, 6(1), 125–137, 2006.

[17] Mech, R.; Prusinkiewicz, P.: Visual models of plants interacting with their environment. 23rd Conference on Computer Graphics and Interactive Techniques, 1996.

[18] Mun, D.; Han, S.: Identification of topological entities and naming mapping for parametric cad model exchanges. International Journal of CAD/CAM, 5(1), 69–81, 2005.

[19] Prusinkiewicz, P.; Hanan, J.: Lindenmayer systems, fractals, and plants, 1989.

[20] Quattrini, R.; Baleani, E.: Theoretical background and historical analysis for 3d reconstruction model : Villa thiene at cicogna. Journal of Cultural Heritage, 16, 119–125, 2015.

[21] Terraz, O.; Guimberteau, G.; Merillou, D., S.and Plemenos; D, G.: 3gmap l-systems: an application to the modelling of wood. Formal Methods in Software and System Modeling, 25(2), 165–180, 2009. http://doi.org/10.1007/s00371-008-0212-5.

[22] Wang, Y.; Nnaji, B.: Geometry-based semantic id for persistent and interoperable reference in feature-based parametric modeling. Computer Aided Design, 37(10), 1080–1093, 2005. http://doi.org/10.1016/j.cad.2004.11.009.

[23] Wu, J.; Zhang, T.; Zhang, X.; J., Z.: A face based mechanism for naming, recording, and retrieving topological entities. Computer-Aided Design, 33(10), 687–698, 2001.

[24] Xue-Yao, G.; Jia-Qi, L.; Hao, G.; Yun-Feng, G.: Name and maintain topological faces in rotating and scanning features. International Journal of Grid and Distributed Computing, 9(3), 21–26, 2016.