Title:
Persistent Naming Based on Graph Transformation Rules to Reevaluate Parametric Specification

Authors:
Anaïs Cardot, anais.cardot@univ-poitiers.fr, University of Poitiers
David Marcheix, david.marcheix@ensma.fr, ENSMA
Xavier Skapin, xavier.skapin@univ-poitiers.fr, University of Poitiers

Keywords:
Persistent Naming, Parametric Specification, Rule-based Language, Topological Model

DOI: 10.14733/cadconfP.2018.387-391

Introduction:
CAD softwares are widely based on parametric specifications, which record operations used during the modeling process. Any specification edition results in new models after reevaluation. We address the issue known as persistent naming problem: the ability to find the right parameters and to reevaluate correctly each operation, despite the modeling object modifications. As an example, let a specification made of three operations be described in Fig. 1a: a cube is created, then a slot is applied (slicing face $f$ into $f_1$ and $f_2$), followed by a slot inside $f_2$. Then, we shorten the first slot width then triggers the reevaluation (Fig. 1b): after the cube creation, its front face has not been sliced anymore, so face $f_2$ does not exist. The issue boils down to assign a name to each entity used in the initial evaluation in order to match it with another entity during reevaluation ($f_x$ in this example). Most earlier works [2][3][4] separate 2D from 3D processes and persistent naming solutions are entity dimension-specific. Some approach also record the history of every model entity, instead of keeping track of the ones used as operations parameter only. Finally, no solution has been presented to deal with every kind of specification edition.

Main idea:
We propose a persistent naming system independent of the model dimension, the dimension of entities (vertices, edges and so on) and withstanding the edition of the parametric specification (i.e. adding, deleting or moving operations). Our work relies on the modeling library Jerboa [2], based on the G-Map topological model [5]. This software describes any modeling operation as a transformation rule applied

Fig. 1: Persistent Naming Problem; (a) Initial specification; (b) Reevaluation
on a graph and prevents from any topological inconsistency. We detail how we have the mechanisms of parametric systems inside Jerboa.

G-Maps:
The \(d\)-dimensional G-Map model represents an object as a graph: nodes are called darts and each entity is defined as a specific set of darts linked by relationships denoted as \(\alpha_i\) (\(0 \leq i \leq d\)). Constraints are defined on \(\alpha_i\) to ensure that objects represented as G-Maps are quasi-manifolds. An example is shown in Fig. 2: (a) Geometric representation of a 2D object made of two faces; (b) Faces are separated from each other and are connected by an \(\alpha_2\) arc (blue line); (c) Face edges are separated and \(\alpha_1\) arcs (red lines) link their common vertices; (d) Edges are split and their ends are connected by \(\alpha_0\); (e) those ends form the graph nodes (darts), labeled \(a, \ldots, n\). This graph representation extends seamlessly in any dimension. An entity is represented by a single dart and a specific combination of \(\alpha_i\), for example, the face (resp. edge, vertex) related to dart \(e\) is made of all darts starting with \(a\) and reachable using \(\alpha_0\) and \(\alpha_1\) (resp. \(\alpha_0\) and \(\alpha_2\), \(\alpha_1\) and \(\alpha_2\)) to produce the set of darts \(\{a, \ldots, f\}\) (resp. \(\{e, d, h, g\}, \{e, f, g, n\}\)). In the following, the list of \(\alpha_i, \ldots, \alpha_j\) (\(0 \leq i, j \leq n\)) used to form an entity is called orbit type and is represented as \(\langle i \ldots j \rangle\).

For instance, the orbit type defining a 2D face is \(\langle 01 \rangle\). More generally, for any \(d\)-dim, topological model, there are \((d+1)^2\) different orbit types, including the one designing a single dart and denoted as \(\langle \rangle\); in 2D, those orbit types are \(\langle \rangle, \langle 0 \rangle, \langle 1 \rangle, \langle 2 \rangle, \langle 01 \rangle, \langle 02 \rangle, \langle 12 \rangle, \langle 012 \rangle\).

**Fig. 2:** G-Map representation of 2D objects; Only \(\alpha_1\) relationships connecting pairs of different darts are represented.

Graph Transformation Rules:
A graph transformation rule takes a G-Map as input and produces another G-map as output. Rules defined in Jerboa are graphically described as two patterns made of nodes separated by a left-to-right arrow (see Fig. 3a-b). The pattern on the left is to be filtered in the input G-Map, to identify every part which matches this pattern. The pattern on the right describes the transformation to apply on each of those matching parts, in order to produce the output G-Map. Fig. 3a shows the rule creates a triangular face from scratch: nodes \(m_0\) to \(m_5\) are generated from scratch and are linked using \(\alpha_0\) and \(\alpha_1\) to create a face as the one shown in Fig. 3c. In Fig. 3b, the left pattern is made of node \(n_0\) associated with the orbit type \(\langle 01 \rangle\). Filtering any G-Map using this pattern returns all sets of darts linked by \(\alpha_0\) or \(\alpha_1\). The right pattern is made of three pairs (node, orbit type): \(n_0\) is copied in triplicate to produces nodes \(m_0, m_1\) and \(m_2\) (in Fig. 3d, each dart \(f_0, \ldots, f_5\) has been triplicated in \((a_0, a_1, a_2), \ldots, (f_0, f_1, f_2))\), forming the darts of the output G-Map. Regarding \(m_0\), its orbit type \(\langle 0 \_ \rangle\) means that \(m_0\) keeps the same relationship \(\alpha_0\) as \(n_0\), but \(\alpha_1\) is deleted. Regarding \(m_1\), \(\langle \_2 \rangle\) means that the initial \(\alpha_0\) is deleted and the initial \(\alpha_1\) is replaced with \(\alpha_2\). We proceed the same regarding \(m_2\), Fig. 3e shows the state of the output G-Map at
this stage. Next, relationships $\alpha_0$ and $\alpha_1$ between $m_0$, $m_2$ and $m_2$ are mirrored in the output G-Map (Fig. 3f): the initial face has been subdivided in three sub-faces in a consistent way. Note that this rule can be applied on any 2D polygonal face.

![Graph transformation rule](Fig. 3: Graph transformation rule; (a) TriangleCreation rule; (b) FaceTriangulation rule; (c) Input G-Map; (d) Darts of the output face; (e) First phase of $\alpha_i$ setup; (f) Second phase and output G-Map)

**Persistent Naming System:**

Persistent naming focuses on assigning an identifier to entities used as parameters of modeling operations, during the initial evaluation. This identifier should allow the matching between those entities and the right ones available in the reevaluation. Our naming system is based on darts: each dart of the entity upon which a rule is applied is assigned a Persistent Id (or PId). Each PId is composed of one or several pairs RuleNum-NodeNum, where RuleNum is the number of the rule in the initial evaluation and NodeNum is the node instance in the right part of the rule. As an example, a specification composed of two rules is shown in Fig. 3a-b: \{1-TriangleCreation; 2-FaceTriangulation\}. First rule generates nodes $m_0$ to $m_5$, so after applying it, the output G-Map contains darts $a_0$ to $f_0$ (Fig. 3c); their resp. PId are $1-m_0,\ldots,1-m_5$. Second rule is then applied on the face, resulting in the G-Map shown in Fig. 3f. Each dart of this G-Map is associated with one node among $m_0, m_1$ or $m_2$ and its PId is extended accordingly. As an example, darts $a_0, a_1$ and $a_2$ in Fig. 3f are derived from dart $a_0$ in Fig. 3b: they share the same PId prefix $1-m_0$. Then, their PId is extended with specific information related to $m_0, m_1$ or $m_2$: the complete PId of $a_0$ (resp. $a_1, a_2$) is therefore \{1-m_0; 2-m_0\} (resp. \{1-m_0; 2-m_1\}, \{1-m_0; 2-m_2\}). We proceed the same way for each dart of the output G-Map (as an example, PId of darts $f_0, f_1$ and $f_2$ are resp. $1-m_5; 2-m_0, 1-m_5; 2-m_1$ and $1-m_5; 2-m_2$). Next, we define the Persistent Name (or PN) of entities which are used as parameters of operations, as the concatenation of the PId of one of its darts and the orbit type which represents this entity. For instance, the PN of the face related to dart $a_0$ in Fig. 3e is defined as \{1-m_0; 2-m_0\}.(01). Assume dart $a_1$ has been designated as the representative dart related to the face upon which 1-TriangleCreation has been applied. Then the specification’s contents are \{1-TriangleCreation(); 2-FaceTriangulation({1 – m_0; 2 – m_1}.(01))\}.

**Bulletin Boards and History Records:**
It is necessary to trace the evolution of orbit types between successive operations. Different kinds of evolutions are: Creation, Deletion, Merge, Modification, No Change and Split. To trace the evolution after a single operation, each rule is associated with a table called Bulletin Board (BB), and indexed by orbit types. Each table entry contains at least one tree graph with one or several leaves. The tree roots contain nodes of the rule’s left part, combined with a specific orbit type. The tree leaves contain nodes of the rule’s right part, which share the same leaf if they belong to the same orbit type as the entry index. The tree arcs are labeled with the kind of evolution undergone by orbit types. Fig. 4a describes the Bulletin Board entry indexed by \( \langle 01 \rangle \) (2D face orbit) related to the Triangulation rule (Fig. 3b). The tree leaf regroups \( m_0, m_1 \) and \( m_2 \), meaning that darts related to these nodes belong to the same 2D face in the output G-Map. Let \( (a_0, a_1, a_2) \) be those related darts: as shown in Fig. 3f, the face also contains \( f_0, f_1, f_2 \). It follows that all those darts have been generated using darts \( a_0 \) and \( f_0 \) of the input G-Map (Fig. 3c). \( a_0 \) and \( f_0 \) are related to node \( n_0 \); moreover, since they are linked by \( a_0 \), the tree root is defined as \( n_0 \). The tree arc is labeled Split, meaning that the initial face has been split into several three sub-faces resp. generated by initial darts \( (a_0, f_0), (a_1, a_2) \) and \( (f_1, f_2) \). Every entry of the BB is filled in the same way.

![Fig. 4: (a) Entry \( \langle 01 \rangle \) of the FaceTriangulation rule’s Bulletin Board; (b) Result of applying Coloration rule on the face (orbit type \( \langle 01 \rangle \)) related to \( a_1 \); (c) History Record (HR) of this face; Left: \( \langle 01 \rangle \)-indexed entry of FaceTriangulation rule; Right: \( \langle 0 \rangle \)-indexed entry of TriangleCreation rule](image_url)

To trace evolutions along several operations, we used additional structures called History Records, defined for every PN. Let us add a third operation to the specification: 3-Coloration, which is applied on orbit type \( \langle 01 \rangle \) related to dart \( a_1 \) (see Fig. 4b). As a reminder, \( a_1 \)'s PId is \( \{1 - m_0; 2 - m_1 \} \). Therefore, the PN used as parameter of 3-Coloration is \( \{1 - m_0; 2 - m_1 \}; \langle 01 \rangle \). The creation of the HR is achieved backwards, as shown in Fig. 4c, following red arrows: (1) Coloration is applied on \( \langle 01 \rangle \) (related to \( a_1 \)), which is used as an index in the BB related to 2-FaceTriangulation: the BB’s entry is shown in Fig. 4a. (2) Using the last part of \( a_1 \)'s PId \( \{2 - m_1 \} \), we select in this entry the tree which contains \( m_1 \) and (3) we use the orbit type \( \langle 0 \rangle \) defined in the tree root as an index for the BB related to 1-TriangleCreation. The matching entry is shown on the right side of Fig. 4c: nodes \( m_0 \) to \( m_5 \) shown in Fig. 3a are partitioned in three pairs of nodes linked by \( \alpha_0 \). The common root of these pair is \( \emptyset \), meaning that have been created from scratch. (4) The first part of \( a_1 \)'s PId \( \{1 - m_0 \} \) is used to select the tree which contains \( m_0 \). The orbit type related to the tree root is \( \emptyset \), meaning that there is no previous operation.

Reevaluation:
To handle any edition of the initial specification, we create a matching graph for every parameter of interest, and we rely on its HR to determine how it is affected by the edition. As an example, we insert inside the specification the operation 1.1-VertexInsertion, between 1-TriangleCreation and 2-FaceTriangulation. Fig. 5a-d show the successive steps of the reevaluation. The PN build from \( a_1 \)'s PId and used as pa-
rameter for 3-\textit{Coloration} is \(\{1 - m_0; 2 - m_1\}\). We use its \(HR\) and complete the \textit{matching graph} at each step of the reevaluation to take into account 1.1-\textit{VertexInsertion}. The \textit{matching graph} is shown in Fig. 5e; note that although the \(HR\) has been created in reverse order, we follow it now in the correct order. (1) 1-\textit{TriangleCreation} is reevaluated and a triangle identical to the initial one is created (Fig. 5a). The beginning of the \(HR\) (the \(\langle 0 \rangle\)-indexed entry of \textit{TriangleCreation} rule in Fig. 4c) indicates \(1 - m_0\), so we consider \(p_0, \langle 0 \rangle\) (Fig. 5e), the orbit based on the dart generated from node \(m_0\) of the \textit{TriangleCreation} rule. (2) 1.1-\textit{VertexInsertion} is applied (Fig. 5b). The \(BB\) related to (not shown here) \textit{VertexInsertion} rule indicates that there is a split for orbit type \(\langle 0 \rangle\); indeed, the edge composed of \((p_0, u_0)\) has been split into distinct edges made of \((p_0, w_0)\) and \((u_0, v_0)\). Therefore, in the \textit{matching graph}, two arcs labeled \textit{Split} start from \(p_0, \langle 0 \rangle\) and point to one representative dart of each edge: we choose \(p_0\) and \(u_0\), still combined with \(\langle 0 \rangle\). (3) 2-\textit{FaceTriangulation} is reevaluated (Fig. 5c). This time, similarly to first step, the entry related to \(2 - m_1\) in the \(BB\)'s indexed \(\langle 01 \rangle\) (left part of Fig. 4c), leads to darts \(p_1\) and \(u_1\) generated by node \(m_1\) of the \textit{FaceTriangulation} rule: the \textit{matching graph} is updated with an arc labeled \textit{Split}, starting from \(p_0\) (resp. \(u_0\)) and ending on \(p_1\) (resp. \(u_1\)). (4) 3-\textit{Coloration} is applied on each \textit{matching graph}'s leaf, leading to the coloring of faces containing \(p_1\) and \(u_1\), as intended.

![Fig. 5: Specification reevaluation; (a-d) Successive steps; (e) matching graph](image)

**Conclusion:**
Our approach allows to reevaluate parametric specifications, regardless of the model dimension. The only kind of edition described is adding operation into specifications, but our method manages removing or changing operation order as well. Some development work on the \textit{Jerboa} library is still required to test more complex specifications.

**References:**


