Y. Ikeda, C. M. Herr, D. Holzer, S. Kaijima, M. J. Kim. M, A, Schnabel (eds.), *Emerging Experience in Past, Present and Future of Digital Architecture, Proceedings of the 20th International Conference of the Association for Computer-Aided Architectural Design Research in Asia CAADRIA* 2015, 157–166. © 2015, The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), Hong Kong

TYPES OF PARAMETRIC MODELLING

PATRICK JANSSEN[1] and RUDI STOUFFS[2]
[1,2] *National University of Singapore, Singapore*
[2] *Delft University of Technology, Delft, The Netherlands*
[1] *patrick@janssen.name* [2] *stouffs@nus.edu.sg*

Abstract. Parametric modelling is a term widely used to describe a range of modelling approaches. This paper proposes a taxonomy that distinguishes types of parametric modelling in the way they support iteration. A generalized parametric model is described and used as an analytical device to investigate how different parametric modelling methods provide for iteration over list structures.

Keywords. Parametric modelling, iteration, taxonomy.

1. Introduction

Parametric modelling is a term widely used to describe a range of modelling approaches. There have been previous attempts at classifying parametric modelling methods. Barrios Hernandez (2006) distinguishes parametric variations from parametric combinations (and parametric hybrid models). From a systems point of view, this distinction is uninformative as almost all currently available systems exhibit a hybrid approach in order to provide specific features and capabilities of interest to the users. Monedero (2000) distinguishes variants programming, history-based (constraint) modelling, variational geometry, rule-based variants, and parametric feature-based design. Both variants programming and rule-based variants use a programming approach, the former imperative programming, the latter a form of logical programming. Variational geometry emphasizes the use of a constraint solver; parametric feature-based emphasizes the consideration of a concept of features. Most parametric systems used in architecture would be classified as history-based modelling.

Instead, we adopt a taxonomy that distinguishes types of parametric modelling, based on the way they support iteration. This taxonomy offers a clear and unambiguous way to classify different parametric modelling methods as well as the systems that implement these methods, without lumping most methods within the same category. The paper first describes a generalized parametric model to be used as an analytical device to investigate different parametric modelling methods. In general, a parametric model consists of a collection of modelling operations that are linked into a network that can be topologically sorted, that is, the order of execution of the modeling operations can be defined prior to execution. Therefore, we adopt a Directed Acyclic Graph (DAG) as a generalized representation of a parametric model. Next, we consider a taxonomy of parametric modelling methods based on different iteration methods within a graph, resulting in four types: object modelling, associative modelling, dataflow modelling, and procedural modelling. The final discussion relates parametric modelling to generative modelling using imperative programming.

2. Generalized Parametric Model

We propose a Generalized Parametric Model (GPM), represented by a Directed Acyclic Graph (DAG), as a means to analyse and compare different parametric modelling methods. The GPM is used as an analytical device and there is no suggestion that the systems being discussed were actually implemented using such a DAG. Modelling methods that cannot be mapped into the proposed GPM are categorised as not being parametric modelling methods. This broader category will be addressed in the final discussion.

A GPM graph consists of a set of nodes connected with directed edges. Nodes are distinguished as operation nodes and data nodes. The former represent general computational operations, both geometric and non-geometric. The latter represent the input and output data for the operations, either geometric, non-geometric or a combination thereof. Edges connect operation nodes with data nodes and represent the flow of data from and to the operations. An example of a GPM graph is shown in Figure 1.

Operation Nodes

Operation nodes are typically implemented using an imperative programming language and, as such, can execute any complex procedure. The functionality of an operation node is constrained by the functions that can be invoked in the underlying modelling engine. Modelling engines may focus on different modelling techniques, for example, spline-based modelling, polygon-based modelling, and solid modelling. Some systems also include as operations advanced solvers that take problem descriptions as inputs and use iterative procedures to attempt to calculate a solution. Examples of solvers include particle solvers, rigid body solvers, constraint solvers, and optimization solvers. This paper does not take into consideration these differences in modelling engines and instead focuses on the topology of the DAGs that can be defined.

Each operation node can have multiple inputs and outputs. The inputs may include a set of parameters required for the operation, e.g., an 'extrude' operation may require a list of polygons as input data, as well as a direction vector and the extrusion distance.

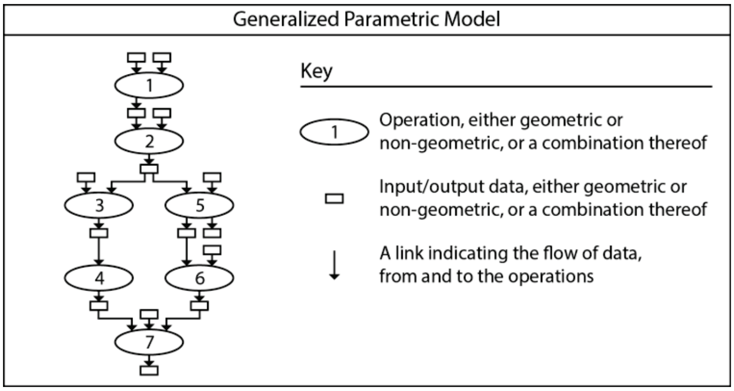
Generalized Parametric Model
1
2
3
5
4
6
7
Key
1
Operation, either geometric or non-geometric, or a combination thereof
Input/output data, either geometric or non-geometric, or a combination thereof
A link indicating the flow of data, from and to the operations

Figure 1: An example of a GPM graph.

Data Nodes

A data node may serve both as an output of one operation and as an input of another operation. The representational data structure used for data nodes may vary depending on the system. Three commonly used data structures are flat lists, nested lists (or multi-dimensional arrays), and topological data structures (such as hierarchical data (tree) structures). In some systems, the user may have no control over the data structure while in other systems users may be provided with operations and tools that enable them to construct customized data structures.

Edges

Edges represent the flow of data, connecting operation nodes with data nodes, and vice versa. The (directed) edges going into an operation represent the data sets consumed by this operation; the edge going out of an operation represents the data set produced by this operation. A data node with more than one edge leaving from it represents a data cloning operation; the respective inputs will be exact copies of each other.

Execution

The execution of a GPM graph is assumed to be performed in a synchronous manner (Lee and Messerschmitt, 1987), with the order of execution defined by applying a topological sort algorithm to the graph. For any set of nodes, many valid orderings are possible, e.g., in Figure 1, the numbering of the operation nodes indicates one possible ordering. Each time the graph is executed, the output data sets are reproduced. Changing the input data will trigger the re-execution of the graph, thereby generating new output data. In most systems, only operation nodes downstream of the changed data will be re-executed.

Iteration

Due to the acyclic nature of the GPM graph, loops cannot be defined. However, this does not rule out iteration over lists of entities. Three broad types of iteration are defined: single-operation iteration, implicit multi-operation iteration, and explicit multi-operation iteration (Figure 2).

The simplest type of iteration is an iteration that applies the same operation simultaneously over multiple geometric entities. For example, if the input of an ‘extrude’ operation consists of a list of polygons, then the node may iterate over the list and extrude each polygon in turn. If the operation takes additional parameters, these parameters would all have a single input value. This type of iteration is referred to as single-operation iteration.

The iteration becomes more complex if additional parameters may also be assigned multiple input values. For example, if the extrusion distances are also provided as a list, then the operation may iterate over both lists, performing some more complex type of data matching. This type of iteration is referred to as implicit multi-operation iteration. In general, it allows for the use of custom data structures consisting of nested lists in combination with data matching algorithms that appropriately interpret these nested lists (Figure 2, top). The user must ensure that the data is structured in an appropriate manner in order to achieve the desired iterative behaviour.

Explicit multi-operation iteration explicitly represents the iterative process using additional nodes with specialized semantics that modify the control flow. In current modelling systems, this is implemented in two ways: using data sinks or using recursion. With data sinks, two nodes with specialized semantics are required: a ‘for each’ operation node iterates over a list and extracts one data item from the list at a time; a ‘sink’ data node collects the results from the application of one or more operations to each data item (Figure

2, middle). When all items in the list have been processed, the ‘for each’ node will trigger the ‘sink’ node, allowing downstream operation nodes to be executed. This approach also allows ‘for each’ nodes to be nested. For example, one ‘for each’ node may iterate over a list of polygons, and a second may then iterate over the list of points in each polygon.

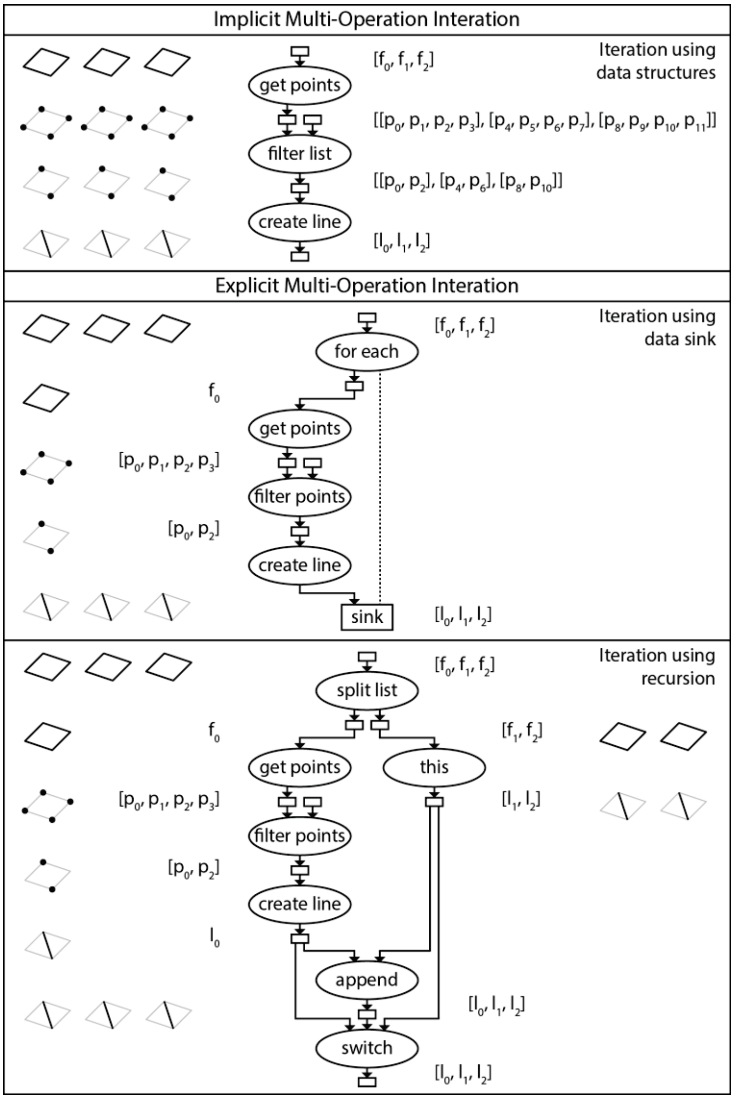

Figure 2: Three different approaches for multi-operation iteration.

Explicit multi-operation iteration using recursion requires just one node with specialized semantics: a node that represents the current subgraph, called 'this' node. When data is input into 'this' node, it is equivalent to re-executing the whole subgraph with new data. A recursive iterator splits an input list into a head and a tail using a 'split' operator (Figure 2, bottom). The head consists of a single data item, to which one or more operations are applied. The tail is a list containing the remaining data items, which is input into 'this' node. Finally, the output from the multiple operations is prepended to the result from 'this' node. Note that a 'switch' operation is required in order to deal with the case when the tail is an empty list.

3. A taxonomy of parametric modelling methods

A taxonomy is proposed that divides parametric modelling into four broad categories, labelled as 'object modelling', 'associative modelling', 'dataflow modelling', and 'procedural modelling'. The distinguishing factor for these modelling methods is how they support iteration. Object modelling does not support iteration and the graph is only implicitly defined. Associative modelling is defined as supporting single-operation iteration, dataflow modelling as supporting implicit multi-operation iteration, and procedural modelling as supporting explicit multi-operation iteration.

Current parametric modelling systems support these three types of iteration to varying extents. Systems that allow the user to directly construct and manipulate the dependency graph are the most powerful. Such graph-based systems include Bentley's GenerativeComponents and Rhino Grasshopper. Both of these systems support implicit multi-operation iteration using nested list data structures. Operations then iterate on the data in the nested lists using various data matching algorithms. Graph-based systems that additionally support explicit multi-operation iteration include Sidefx Houdini and Autodesk Dynamo. Both these systems support explicit multi-operation iteration. Houdini supports iteration using data sinks. Dynamo supports iteration using recursion.

Scene-based systems and feature-based system allow the user to manipulate the dependency graph via various intermediary representations. These types of systems support single-operation iteration, but not multi-operation iteration. Scene-based systems have mainly been developed to support the animation and movie industries. Examples include Autodesk Maya and Autodesk 3DS Max. Feature-based systems have mainly been developed to support mechanical engineering. Examples include Dassault Solidworks, Dassault Catia, and Autodesk Inventor.

More basic types of systems do not support iteration. Trimble SketchUp's 'dynamic components' exemplifies this object modelling approach. Dynamic components are groups of geometries with parameters (and inputs) defined. Operations can be added to repeat a part of a component, to add behaviour to a part or to define a spatial relationship between parts.

3.2. ASSOCIATIVE MODELLING

Graph-based systems are more closely aligned with the proposed GPM and the mapping from these systems to the GPM is relatively straightforward. However, for scene-based and feature-based systems, this is not the case. This section will focus on how the associative representations used in these types of systems can be mapped to GPM graphs.

3.2.1. Scene-Based Systems

Scene-based systems enable users to create scenes populated with objects. Objects are defined using sequences of modelling operations, referred to variously as 'modifier stacks' and 'dependency graphs'. Figure 3 shows an example of a scene-based model on the left, and the equivalent GPM graph on the right.

When using scene-based systems, the two main modelling tasks are creating individual objects and creating the object scene hierarchy. The latter consists of a hierarchical tree of geometric objects that are located in space using associated transformations, including translation, rotation, and scaling. Objects inherit the transformations of their parents.

Each individual object is created using a sequence of modelling operations that are independent from the scene hierarchy. These sequences of modelling operations may also be linked to one another, thereby creating a dependency graph. The order of object creation in the dependency graph may differ completely from the order of the objects in the scene hierarchy

Upon mapping the dependency graph into a corresponding GPM graph (Figure 3), the associated transformations from the scene hierarchy are added as operation nodes. The transformation parameters are mapped into data nodes providing inputs to these operations, in such a way as to replicate the object relationships defined in the scene hierarchy.

3.2.2 Feature-Based Systems

Feature-based systems enable users to create parametric models consisting of assemblies of parts. The parts are defined using feature trees, where each feature represents a modelling operation. Figure 4 shows an example of a feature-based model on the left, and the equivalent GPM graph on the right.

When using feature-based systems, the two main tasks are creating individual parts and creating assemblies of the parts. In an assembly, parts are located by defining relationships with other parts, where relationships consist of constraints and joints. A 3D solver is then used in order to search for configurations that satisfy these relationships.

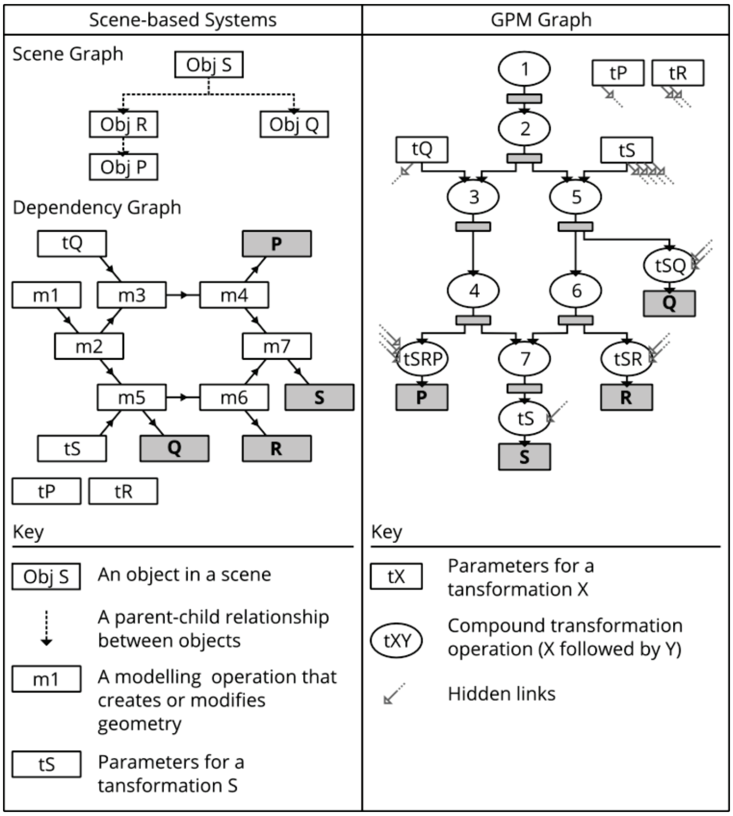
Scene-based Systems
GPM Graph
Scene Graph
Obj S
Obj R
Obj Q
Obj P
Dependency Graph
tQ
P
m1
m3
m4
m2
m7
m5
m6
S
tS
Q
R
tP
tR
Key
Obj S
An object in a scene
A parent-child relationship between objects
m1
A modelling operation that creates or modifies geometry
tS
Parameters for a tansformation S
1
2
tP
tR
tQ
tS
3
5
tSQ
Q
4
6
tSRP
7
tSR
P
R
tS
S
Key
tX
Parameters for a tansformation X
tXY
Compound transformation operation (X followed by Y)
Hidden links

Figure 3: An example model from a (general) scene-based system and the corresponding GPM graph.

Each individual part is created using a sequence of modelling operations, or features. Typically, three types of features can be defined: sketched features, placed features, and work features. Sketched features are operations that generate geometry from 2D or 3D drawings, called ‘sketches’. These sketches are either linked to one of the planes in the origin coordinate system or are linked to a plane in the geometric model. Sketches can include various constraints, and a solver is used in order to modify the drawing to satisfy the constraints. Placed features are operations that modify the existing geometry in some way. Lastly, work features are operations that create construction geometry that is not included in the final output for the part.

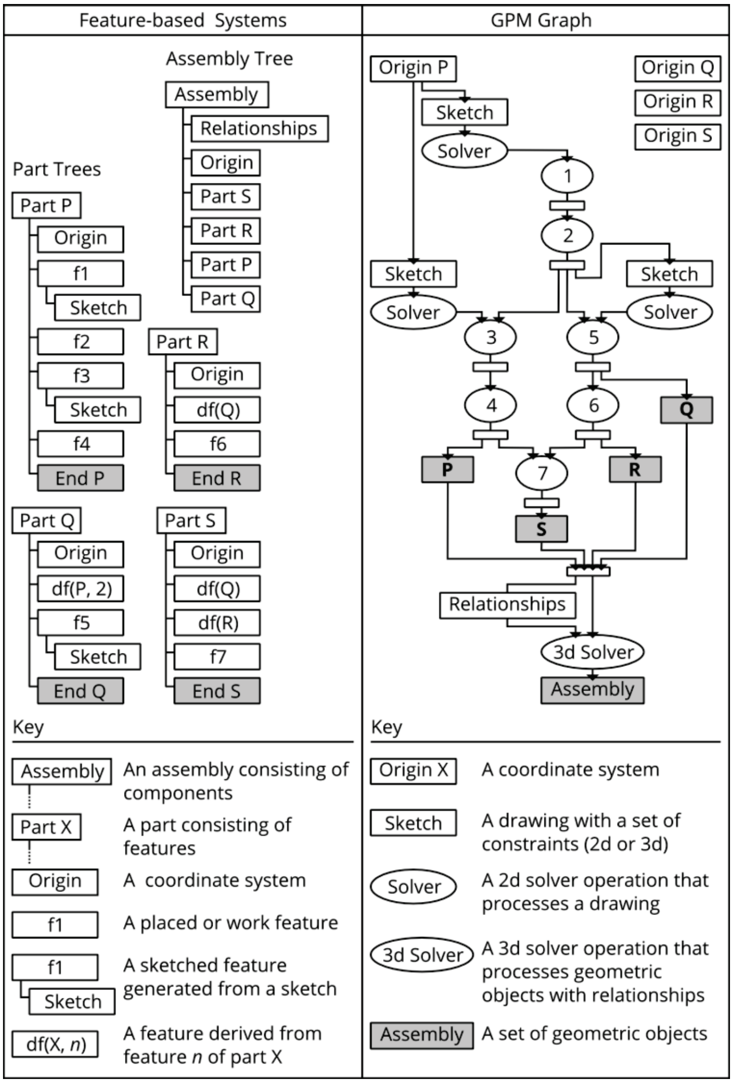

Figure 4: An example model from a (general) feature-based system and the corresponding GPM graph.

Parts can also be linked to one another by creating ‘derived parts’. A derived part is a part that derives some of its geometry from the geometry in another part. This interlinking of parts allows the equivalent of a dependency graph to be created. However, the feature-based systems do not typically provide an explicit representation of this dependency graph; instead, the dependency graph needs to be inferred from the various part trees.

Mapping the assembly tree and the part trees into a corresponding GPM graph (Figure 4) requires the various part trees to be combined into a single graph. For each sketched feature, a data node and solver is inserted into the graph. The assembly tree is then mapped into a set of relationships and a 3D solver in the graph. The relationships define a set of constraints and joints between the geometric objects. The solver is then used to position the objects is such a way so that the relationships are all satisfied.

4. Discussion

We have adopted DAG to represent a Generalized Parametric Model (GPM), as a means to analyse and compare different parametric modelling methods. However, there is at least one type of parametric modelling that we have omitted in our taxonomy, parametric modelling through imperative programming. One reason is that imperative programming cannot be mapped into a GPM graph, simply because imperative programming supports loops and thus cannot be represented through an acyclic graph. Another reason is the fact that imperative programming is inherently parametric (Gürsel Dino, 2012, p. 210) and, as such, casting imperative programming as a parametric modelling method is rather uninformative. Nevertheless, if we were to include it in our taxonomy, an additional category called ‘generative modelling’ could be defined at the same level as parametric modelling.

5. Conclusion

We have adopted a taxonomy based on how parametric modelling methods support iteration, as it offers a clear and unambiguous way to classify different parametric modelling methods as well as the systems that implement these methods, without lumping most methods within the same category.

References

Barrios Hernandez, C. R. (2006), Thinking parametric design: introducing parametric Gaudi, *Design Studies*, **27**(3), 309–324.

Gürsel Dino, İ.. (2012), Creative design exploration by parametric generative systems in architecture, *METU JFA*, **29**(1), 207–224.

Lee, E. A. and Messerschmitt, D. G. (1987), Synchronous data flow, *Proceedings of the IEEE*, **75**(9), 1235–1245.

Monedero, J. (2000), Parametric design: a review and some experiences, *Automation in Construction*, **9**, 369–377.