An Abstract Graph Pattern Collection

CHRISTOPHER HARTLEY, Cisco Systems
NIGEL DAVIS, Ciena Corporation

Software engineering seeks to represent the real-world using models. An important part of this process is choosing the best model to represent a real-world domain, based on the requirements of that domain.

In this paper we discuss the strengths and weaknesses of different modelling approaches for modelling physical and logical networks derived from the concepts of graph theory.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures—Patterns

General Terms: Design

Additional Key Words and Phrases: architecture patterns, abstract graph

ACM Reference Format:

Hartley, C and Davis, N. 2020. An Abstract Graph Pattern Collection. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 27 (October 2020), 21 pages.

1. INTENT

The original intent (around 15 years ago) was to determine what options could be used for the modelling of telecommunications networks. To aid in the production of a 'complete set' of patterns, this was changed to a related problem : "How many ways can an abstract graph be represented in UML ?"
The result was the production of this collection of eight closely related patterns that are variants or alternatives for modelling connectivity.
Because these eight patterns are so closely related, they share common problem and context statements. The forces that indicate which is the best option for a given set of requirements are listed separately in each pattern.

2. CONTEXT

Good representations of network connectivity are essential for the management of telecommunications networks and this has driven the need to research the different possible representations of networks with different characteristics.

There are many models that explicitly or implicitly represent graphs. Most appear to use the same pattern, even though they often have different problem domain requirements. By evaluating the different patterns in this paper, the most appropriate representation can be chosen for each situation.

The use of abstract graph terms also helps to remove any problem domain bias from the patterns that may impede their use in other areas outside of network management.

These patterns may prove useful in many diverse areas, especially where there are complex role based relationships such as :

- The roles people play in a family relationship
- Financial relationships such as parties to a contract
- Representations of business tasks and their relationships
- Physical communications networks, such as optical fiber networks

Note that in the progression of patterns, that later patterns are not 'better' than the earlier ones.
The patterns are in order of sophistication as later ones are developed from the earlier ones.
The patterns deliberately focus on the connectivity aspect of the problem (the Graph Edges) while keeping the things being connected (the Graph Vertexes) unchanged.
The general advice is to use the simplest pattern that meets all of the needs of a given situation.

In network management, connectivity is usually explicit. In other situations, it may not be apparent.

3. PROBLEM

The problem was that in designing network management models, modelling of connectivity was inconsistent, and often ineffective. The understanding of connectivity in each domain was varying the way it was modelled. Focusing on a problem space with no inbuilt bias, an abstract graph, allowed for the basic connectivity forms to emerge and then provide insight back to the network management concepts.

4. FORCES

The forces that determine which option to use are described in each pattern section and then summarized in the conclusion.

5. ABSTRACT GRAPH CONCEPTS

Consider a simple representation of a homogeneous graph made up of Vertexes connected by Edges, which can be directed or undirected, as shown in Figure 1.

Our starting point is that an Edge is assumed to have two ends, an aEnd and a zEnd.

If the Edge is directed, then it is from the aEnd to the zEnd.
If the Edge is undirected, then end representations don't have any distinguishing meaning.

Both directed and undirected Edges are found in real life, based on their symmetry (or lack of it) :

- Optical light signals travel from one point to another (directional flow of photons)
- Tom and Anne are cousins and this is a symmetric relationship (undirected)

As well as the Edge direction symmetry there may be other types of symmetry related to the Edge end attributes which can influence the choice of pattern.

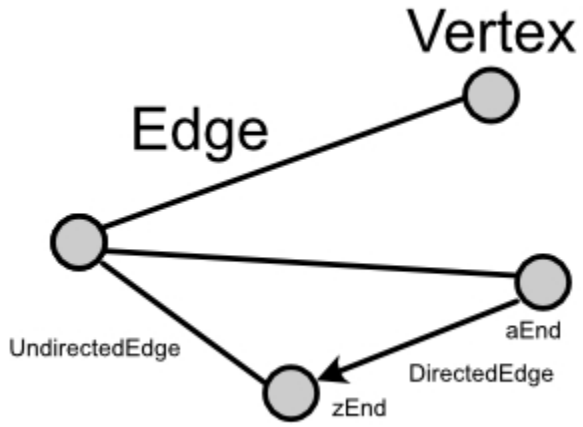

Fig. 1. Abstract Graph.

Some networks have parallel edges (for example for redundancy) and this requirement will influence the choice of pattern for those types of networks.

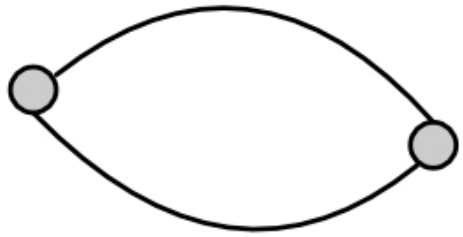

Fig. 2. Abstract graph with parallel Edges

This paper will show the different ways that an abstract graph like that shown in Figure 1 can be represented using UML, starting with the simplest possible representation.

For the UML instance diagram example for each pattern, a simple example of a Phone connected to a USB Hub using a USB cable will be used.

Fig. 3. Phone and Hub

The Phone and Hub examples show that many of these graph pattern alternatives can be applied to a single situation and that it depends on the requirements of the level of detail and what is desired to be in the representation as to the best choice to be used.

To be able to show the examples in the UML tooling, the following attributes have been added to the model to support this particular example and are not part of the actual patterns :

- colour
- name
- connectorType
- terminatedBy

Also note that the attribute ‘name’ is used in the examples as a simple identifier only, and should be replaced with the actual identifier(s) needed for the given situation.

5.1 Edge as Association Pattern

FORCES : Use this pattern to represent the existence or absence of a relationship between two Vertexes and if parallel edges are not required

The first pattern uses a class to represents the graph Vertexes and an association to represent the Edges.

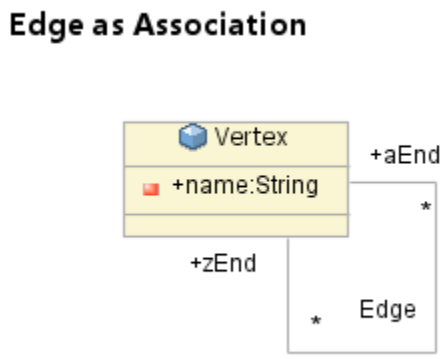

Fig. 4. Edge as Association Pattern

This is the simplest graph representation possible using UML and can really only indicate the presence or absence of a relationship between Vertexes.
Note that it produces a few limitations on the graphs that it can represent. For example, it can't represent a graph with parallel edges, like that shown in Figure 2 because an association can only represent the presence or not of a relationship.
The edge ends are hard-coded into the structure of the pattern (aEnd and zEnd) and so it can not represent edges with more than two ends.
The other key limitations are that it cannot represent attributes for Edges and cannot distinguish between directed and undirected Edges.

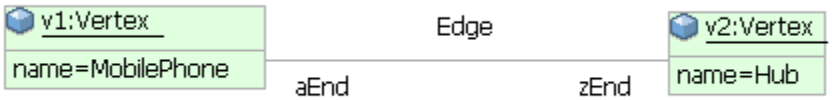

Fig. 5. Edge as Association Instance Example

Self-join associations appear in many models where the connectivity is not really of importance.

5.2 Edge as Association Class Pattern

FORCES : Use this pattern to represent the existence of a relationship with attributes between two Vertexes and parallel edges are not required

The second pattern uses an association class to represent the graph Edges.

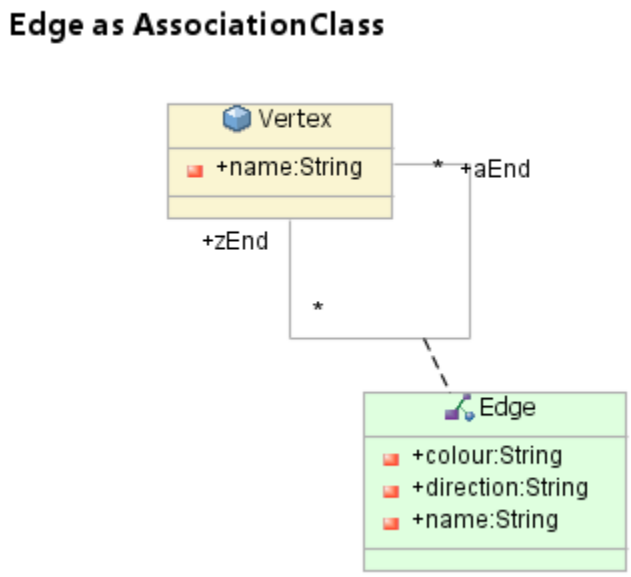

Fig. 6. Edge as Association Class Pattern

By changing the Edge representation from an Association to an Association Class, attributes can now be attached to the Edge representation.
It can now distinguish between bidirectional and unidirectional Edges with the 'direction' attribute.
The inability to represent parallel Edges is still present because the association class identity is still tied to the aEnd and zEnd vertex identities.
The edge ends are still hard-coded into the structure of the pattern (aEnd and zEnd) and so it can not represent edges with more than two ends.

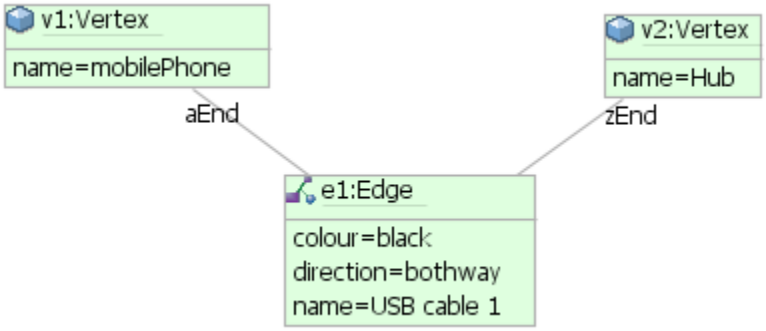

Fig. 7. Edge as Association Class Instance Example

5.3 Edge with dual Associations Pattern

FORCES : Use this pattern to represent a directed two ended relationship with attributes and if parallel edges are required

The third pattern uses a class to represent the Vertexes and a class with two associations to represent the Edges. One association represents the aEnd of an Edge terminating on a Vertex and the other represents the zEnd of an Edge terminating on a Vertex.

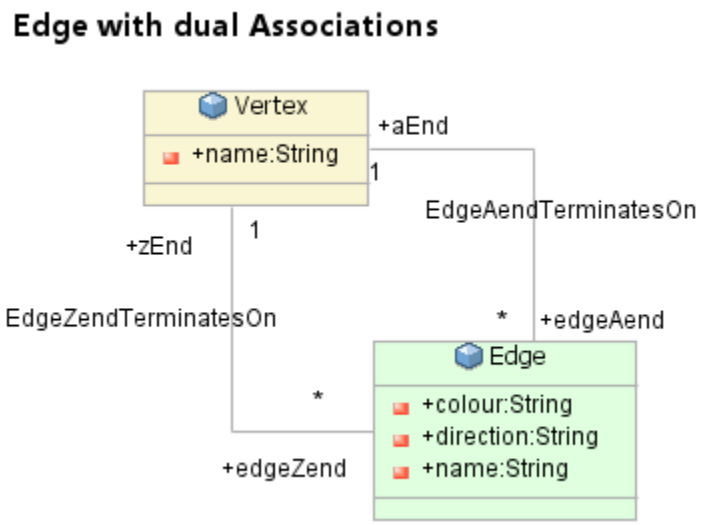

Fig. 8. Edge with dual Associations Pattern

By changing Edge from an Association or Association Class to a class, Edge now has its own identity and this pattern can represent parallel edges, as shown in Figure 2.
The number of Edge ends is still hard-coded into the structure of the pattern and so it can not represent Edges with more than two ends (unless more end associations are added).
A 'direction' attribute can be used to state if the Edge is directed or undirected, but the representation is not consistent.

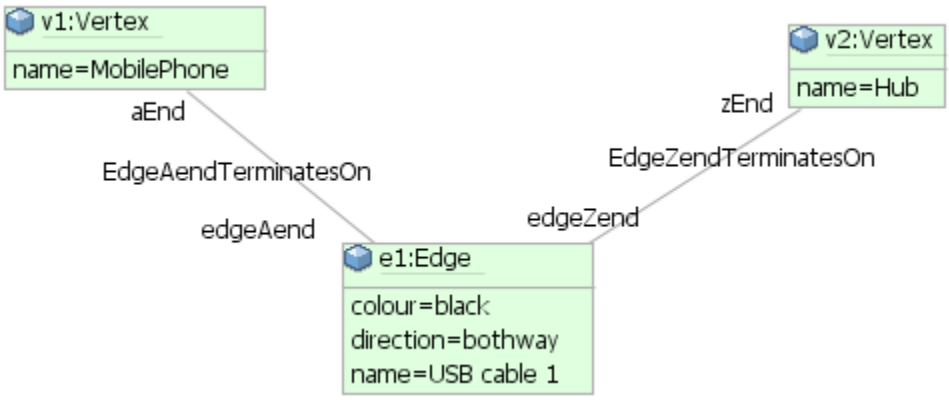

Fig. 9. Edge with dual Associations Instance Example

This is the most common graph representation found in existing models. It is useful where there is a definite direction of point to point flow. For example a normal plane flight leg between airports. It would be less suitable for things like complex flows that can split and combine.

5.4 HyperEdge enhancement to the basic Abstract Graph concepts

The basic graph concepts can be extended by allowing an Edge to have more than two ends, which is called a HyperEdge [Wolfram-Hyperedge].

Figure 10 shows a simple representation of a HyperEdge, as a 'line' with many 'ends'.
Note that this representation attempts to look like a "n-dimensional line". Another option is the set based diagram used in [Wikipedia], but it is less "network-like" and hence not used here.

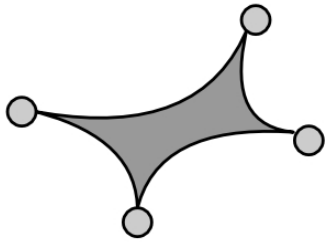

Fig. 10. HyperEdge

When there is an undirected HyperEdge relationship involving n Vertexes, then this could be represented by n.(n-1)/2 two ended relationships. The issue then is that these individual relationships aren't related to each other and for an implementation of a model, the result may be very hard to maintain.

In a similar way to how Figure 2 showed a graph with parallel Edges between a Vertex, Figure 7 shows 'parallel' HyperEdge terminations on a Vertex.
Having a model that supports such cases may prove useful.

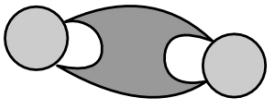

Fig. 11. HyperEdge with parallel terminations

By expanding the concept of Edge to include Edges with more than two ends, a closer semantic mapping can be achieved from the real-world to the model.

For example, if we have a family relationship, then all of the binary relationships in Figure 12, could be converted to an equivalent HyperEdge representation as shown in Figure 13.

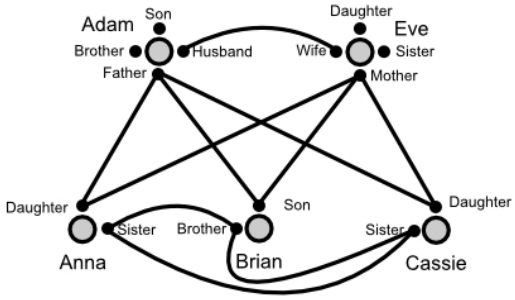

Fig. 12. Family Binary relationships

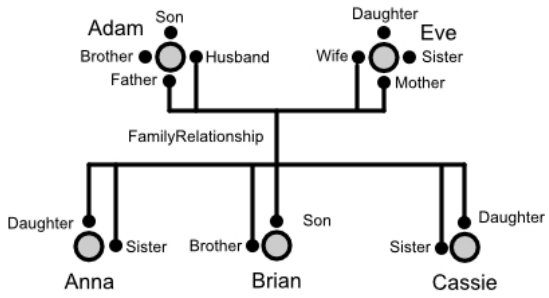

Fig. 13. Family relationships converted to a HyperEdge representation

It may also be possible to normalize the roles to provide a simplified HyperEdge representation as shown in Figure 14.

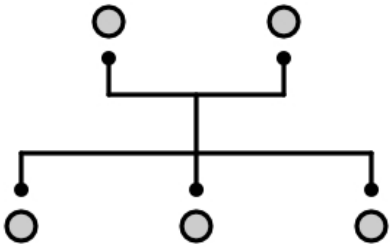

Fig. 14. Normalised HyperEdge relationships

The patterns following this section all provide support for HyperEdges.

5.5 Edge Class Pattern

FORCES : Use this pattern to represent a symmetric multi-ended relationship with attributes and if parallel edges are required

The fourth pattern is a simplification of the previous one, where the two associations are merged into one, and the multiplicity at the Vertex end is changed to *.
The Edge Class pattern is not really useful except in very special limited cases and is just shown to fill in the pattern sequence.

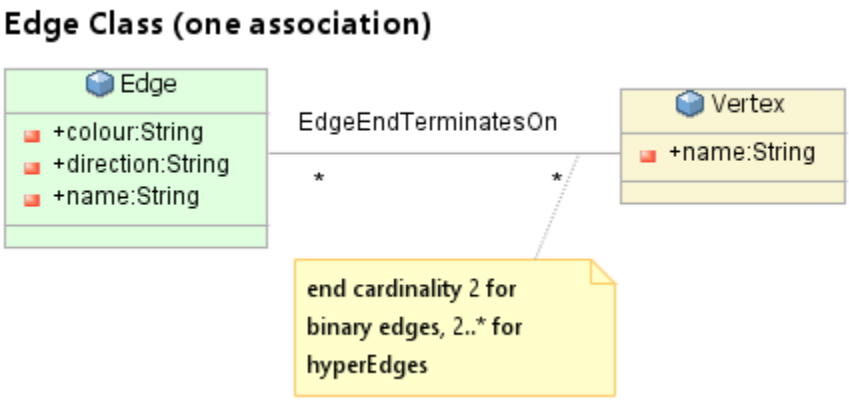

Fig. 15. Edge Class Pattern

This pattern is quite different from the earlier ones, because it is the first pattern that doesn't have aEnd and zEnd association ends terminating on the Vertex class.
The earlier Edge representations are asymmetric due to the fixed aEnd and zEnd association end roles and therefore are natural for representation of directed Edges.
The Edge Class pattern is only useful for undirected Edges as it doesn't have the aEnd and zEnd concepts.

Another item of interest is that the earlier patterns could only represent Edges with two ends and this is the first pattern that can represent HyperEdges.

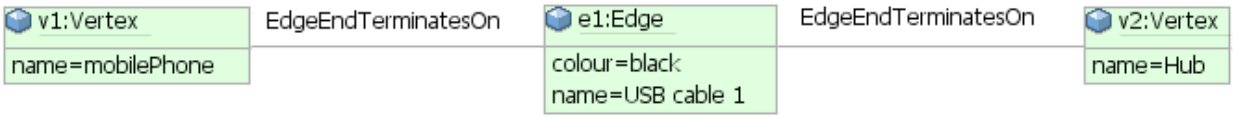

Fig. 16. Edge Class Instance Example

The only practical use for this pattern could be in special cases like the switching matrix of a piece of equipment that supports bidirectional (multi-ended) broadcast connections.

5.6 EdgeEnd Association Class Pattern

FORCES : Use this pattern to represent a symmetric multi-ended relationship with attributes and also with end attributes and if parallel edge end terminations are not required

For the fifth pattern, the association in the Edge Class pattern is converted to an association class.

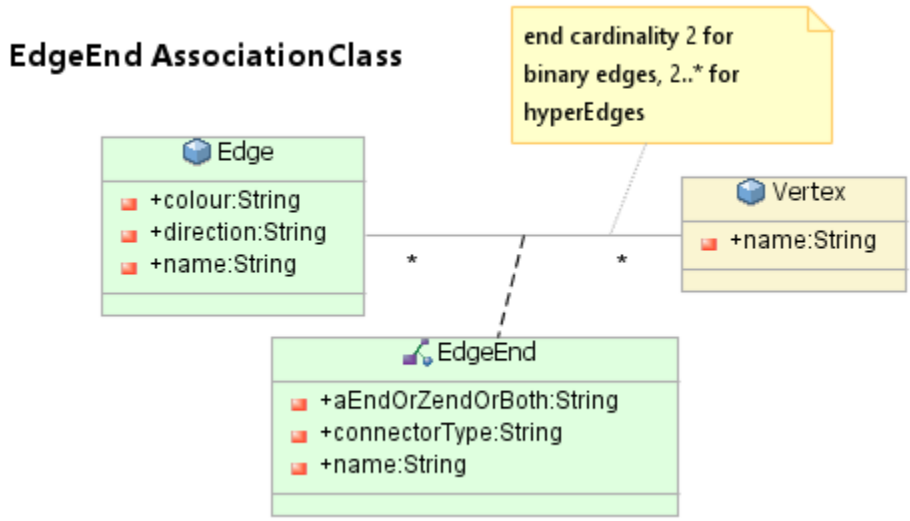

Fig. 17. EdgeEnd Association Class Pattern

This pattern is much more practical than the Edge Class pattern as it can distinguish between directed and undirected edges using an EdgeEnd attribute.
It can also now have attributes relevant to the Edge end such as “connectorType” added here just for the phone and hub example.

This representation does still suffer from a variant of the previous parallel edge issue, as it can’t represent parallel terminations of a HyperEdge on a Vertex (Figure 11).

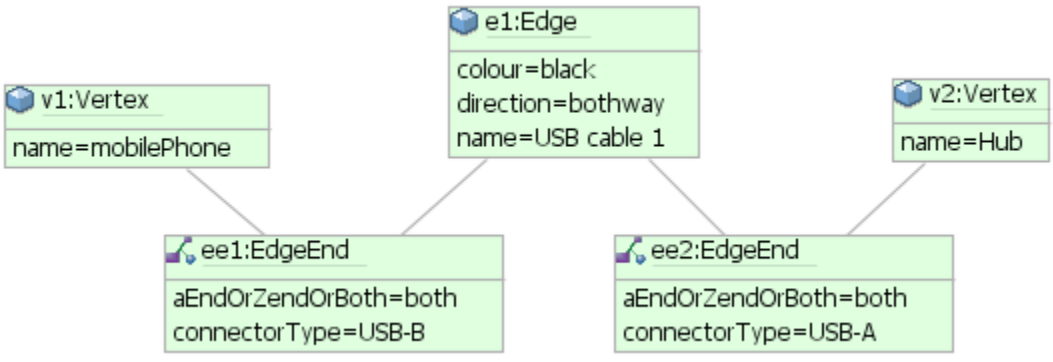

Fig. 18. EdgeEnd Association Class Instance Example

5.7 EdgeEnd Class Pattern

FORCES : Use this pattern to represent an asymmetric multi-ended relationship with attributes and also with end attributes and if parallel edge end terminations are required

The sixth pattern is a simple progression where the association class has been replaced with a class and two associations.

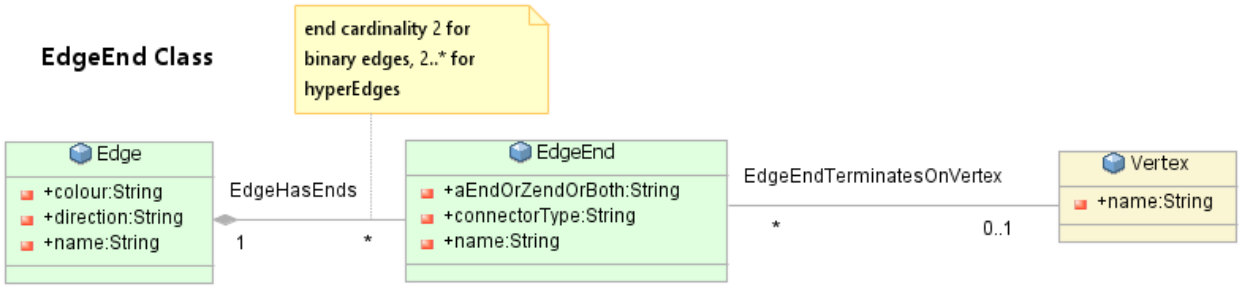

Fig. 19. EdgeEnd Association Class Pattern

This is the first pattern where it is possible to represent Edge ends that exist without terminating on a Vertex.

The independent EdgeEnd representation is an important step as it allows for an unterminated physical conductors to be represented. In fact this is the starting pattern for representations of communication cables. We will define the term 'strand' to refer to an atomic wire or conductor or piece of optic fibre. A cable is then a collection of strands grouped with an outer cover.

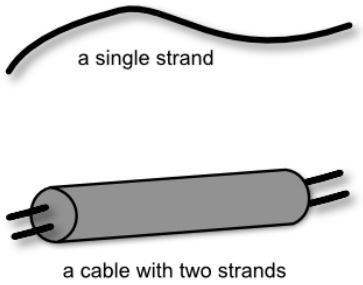

Fig. 20. Strand and Cable

With physical cables and strands, often the properties are unchanging along their length, so that the unchanging values can be added as attributes to Edge and any properties specific to their ends are added to EdgeEnd.

While simple physical cables/strands only have two ends, there are other physical examples of multi-ended connectors. In printed circuit boards there are often connectors connecting multiple points.

In the radio world, broadcast and multi-path reception is also often used, and a HyperEdge provides a good representation of this type of relationship.

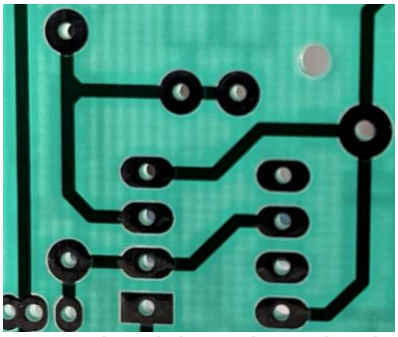

Fig. 21. Multi-ended printed circuit board traces

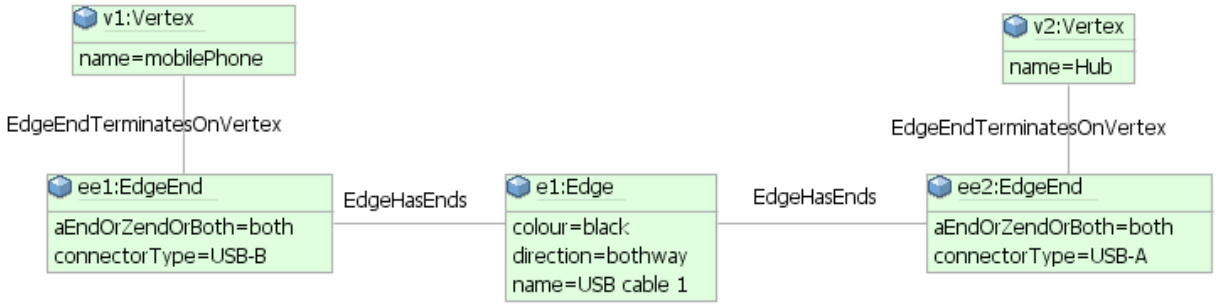

Fig. 22. EdgeEnd Association Class Instance Example

This pattern could also be useful in situations like :

- Multi-legged financial transactions, such as Sales and Contracts ([Fowler] fig 6.3)
- Legal cases
- Manufacturing activities
- Multi-point network architectures such as hub-spoke designs
- Representation of redundancy with main and standby roles
- Physical and logical bus structures (compute bus, publish-subscribe messaging)

5.8 Extension of the Abstract Graph Concepts with Termination

Up to now, standard abstract graph concepts of Edge and Vertex have been used. The advantage of that abstraction is that it can be used in many situations, but the downside is that it may not represent specific real world situations well. The following patterns will extend the basic graph concepts in a way that allows for more useful physical connectivity representations. Branching off the abstract concepts in other ways may be useful for other situations such as logical connectivity.

We will extend the abstract graph concepts as shown in Figure 23, by adding the concept of Termination, where a Termination represents the 'attachment' of the EdgeEnd to a Vertex.

This allows the representation of additional information such as :

- How it is terminated
- When it was terminated
- Who terminated it
- Why it was terminated

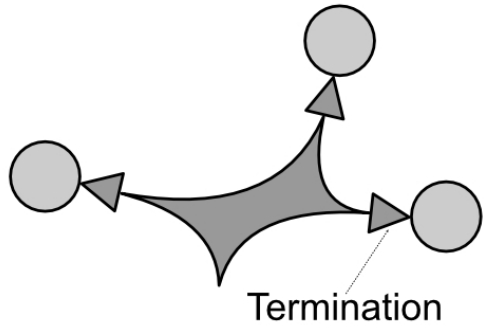

Fig. 23. Edge termination on a Vertex

The Termination concept provides support for physical network processes, where the strands rarely change, and the most common tasks relate to terminating the strand ends.

5.9 EdgeEndTermination Class Pattern

FORCES : Use this pattern to represent an asymmetric multi-ended relationship with attributes and also with end attributes and if parallel edge end terminations with attributes are required

In this pattern the termination of the Edge end on the Vertex is separately modelled and this can be done with either an AssociationClass or a Class.

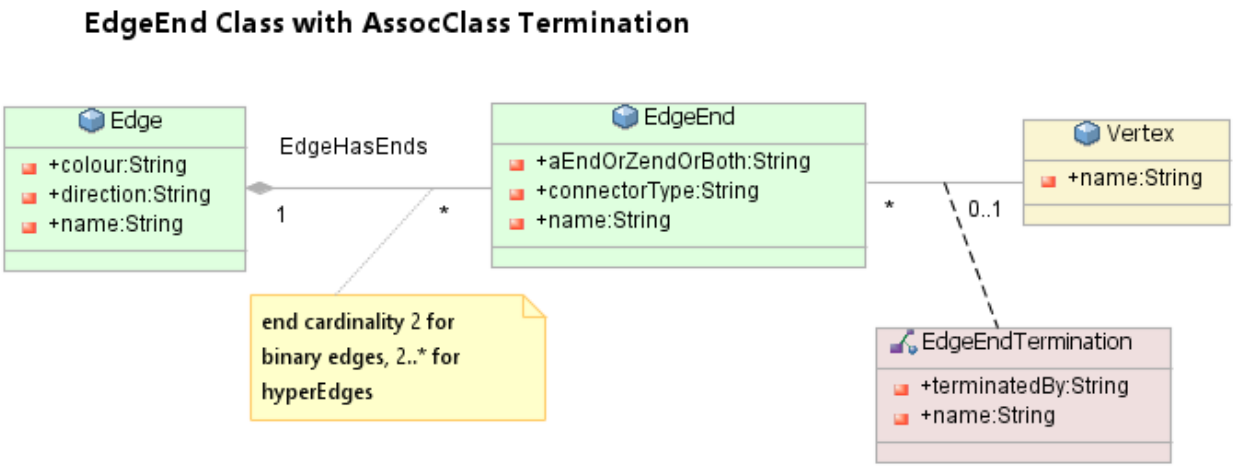

Fig. 24. EdgeEndTermination Association Class Pattern

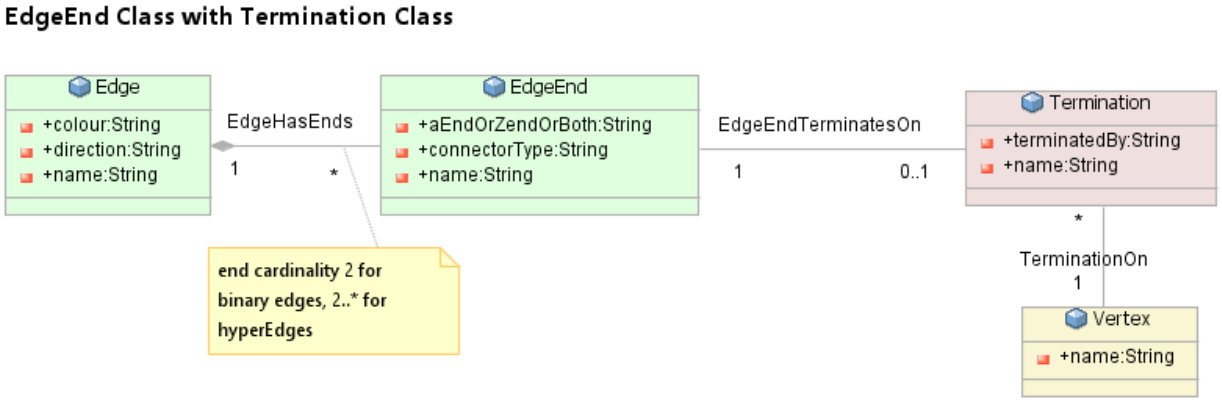

Fig. 25. EdgeEndTermination Class Pattern

This pattern is a useful one for physical connectivity representations such as telecommunications cables as it allows for representing the fusion of fibre strands or the connection of copper pairs to be represented.
It also aligns closely with the soldering points on the printed circuit board discussed previously in Figure 21.

Note that this EdgeEndTermination Class pattern could allow for an EdgeEnd to be terminated on another EdgeEnd if the multiplicity of association EdgeEndTerminatesOn at the EdgeEnd end is changed from '1' to '2' or higher.

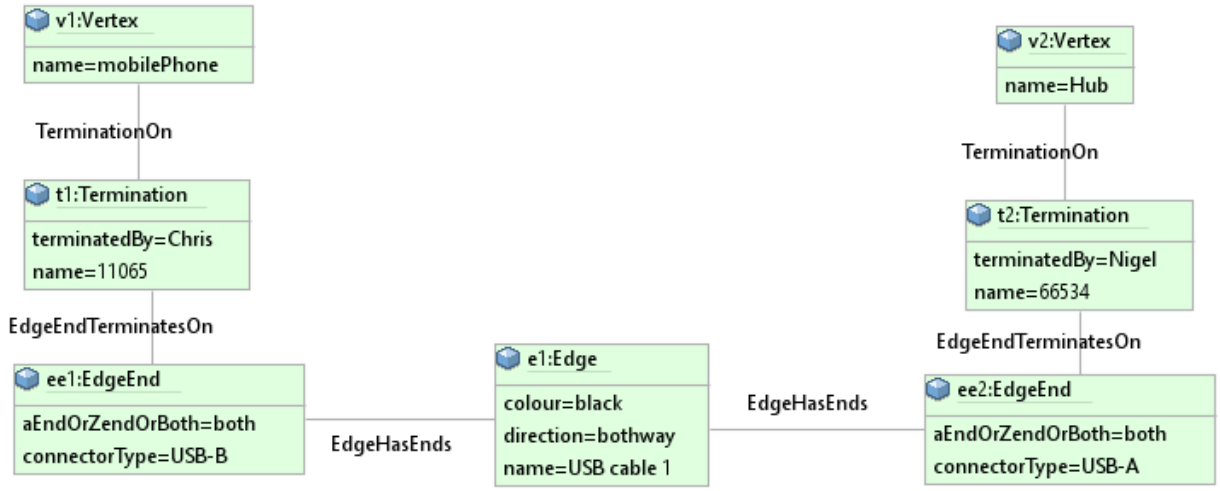

Fig. 26. EdgeEndTermination Class Instance Example

5.10 TerminationEnd Pattern

EdgeEndTermination is the last of the general representations.
One more pattern progression has been found to be useful, but only for a very specific case.

That case is :

- For representation of physical connectivity
- Where the physical connectivity is large and complex, requiring the use of strand and connector pin ranges

For this particular case of representation of large cable networks, adding a TerminationEnd class allows for recording the range numbering mappings on each side of the termination and other more complex terminations found in real world physical networks.

6. CONCLUSION

This paper has presented a number of ways that an abstract graph can be represented in UML, in order of increasing sophistication.

> *"Models are not right or wrong; they are more or less useful."* [Fowler]

The challenge is to pick the most useful pattern for a given situation.

Based on our experience with these patterns :

- For modelling physical connectivity (strands, connectors), 'EdgeEnd class with Termination class' is probably the best option
- For modelling logical connectivity with roles (family, business, legal relationships), 'EdgeEnd class' is probably the best option
- For modelling logical connectivity without roles in simple cases, 'Edge as Association' or 'Edge with dual associations' may be sufficient

The options discussed previously are summarized in the table below.

Option	Multiset graphs (parallel edges)	HyperEdge support	Asymmetric Edge (end roles)	Best for Directed / Undirected
Edge as Association	N	N	N	DIRECTED
Edge as AssociationClass	N	N	N	DIRECTED
Edge with dual Associations	Y	N	N	DIRECTED
Edge class (one assoc)	Y	Y	N	UNDIRECTED
EdgeEnd AssociationClass	Y	Y	Y	BOTH
EdgeEnd Class	Y	Y	Y	BOTH
EdgeEnd class with AssocClass termination	Y	Y	Y	BOTH
EdgeEnd class with Termination class	Y	Y	Y	BOTH

The figure below summarizes the progression through the eight patterns, showing how an association becomes an association class and then a class.
It also shows that the "Edge with Dual Association" pattern is an outlier in the sequence.

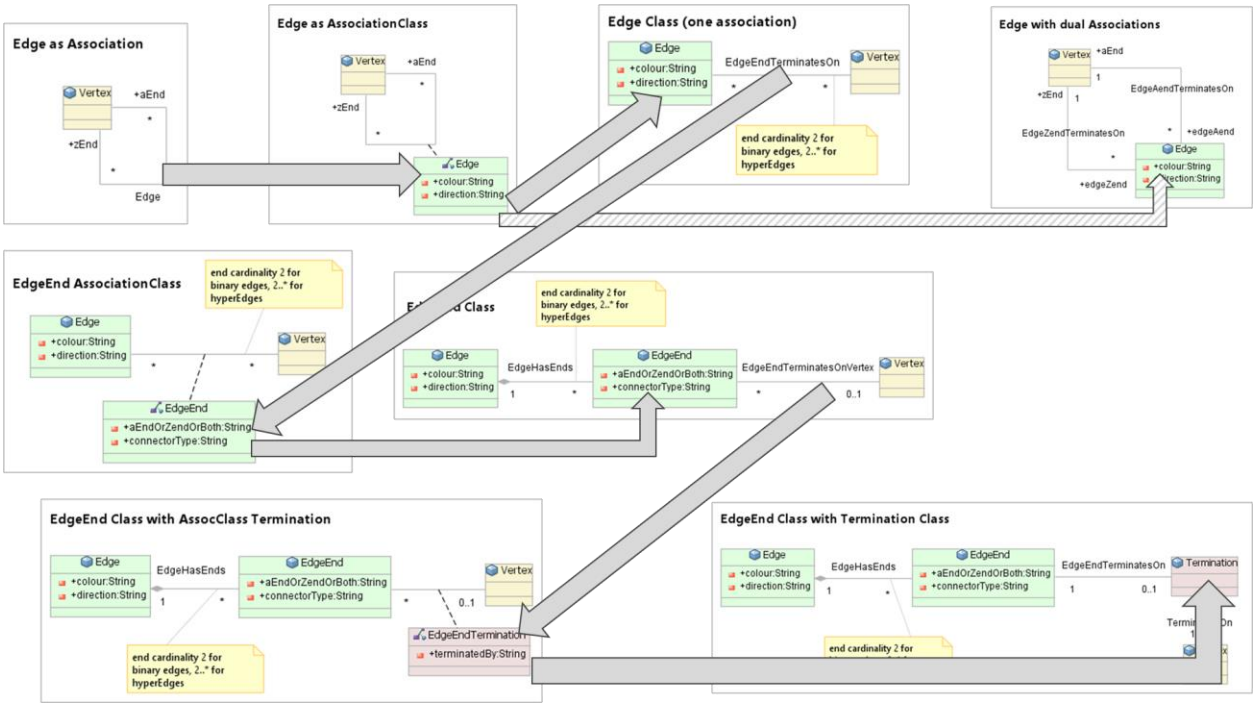

Fig. 27. Pattern Progression

> *"Design is all about trade-offs, and every one of these patterns comes with its own set of trade-offs. Any of the patterns in this paper can be the right pattern for your problem. Dogmatic adherence to one pattern is not the answer, you have to understand an evaluate the trade-offs. This truth is eternal in design."* [Fowler-Roles]

An anti-pattern to watch out for, is models using the 'Edge with dual associations' pattern that have duplicated attributes with names like aEndLocation, zEndLocation, aEndWeight, zEndWeight ….

Many years of experience in representing telecommunication systems using UML has shown that consolidating any end related duplicated attributes and then refactoring to the "EdgeEnd Class" pattern has unlocked our models to provide better representations of graph structured information.

7. APPENDIX A – CONVERTING AN ASSOCIATION CLASS TO A CLASS

In the main text, some of the progressions convert an Association Class to a Class.
This appendix shows how the mapping is done.
Assume that we have an Association Class A, as shown in Figure 28.

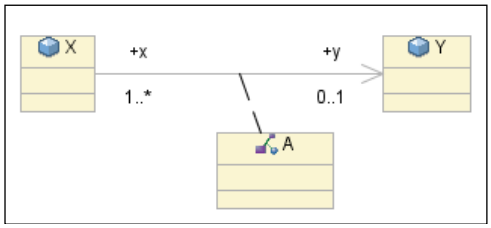

Fig. 28. Association Class

To replace the Association Class with a Class, we need to map it as shown in Figure 29, so that :

- when X looks at class A, it sees the same role, multiplicity and navigability that it did when it looked at Y.
- when Y looks at class A, it sees the same role, multiplicity and navigability that it did when it looked at X.

The multiplicities from class A outwards are always 1 and there is always navigability outwards from class A to both classes X and Y.

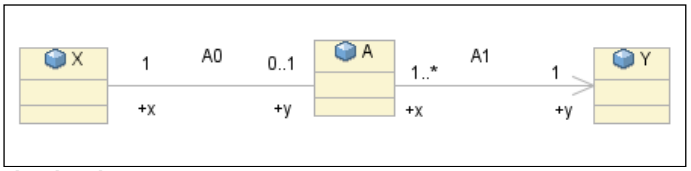

Fig. 29. Association Class replaced with a Class

Another way of thinking about the transformation is to think of the association ends 'crossing over A', as shown in Figure 30.

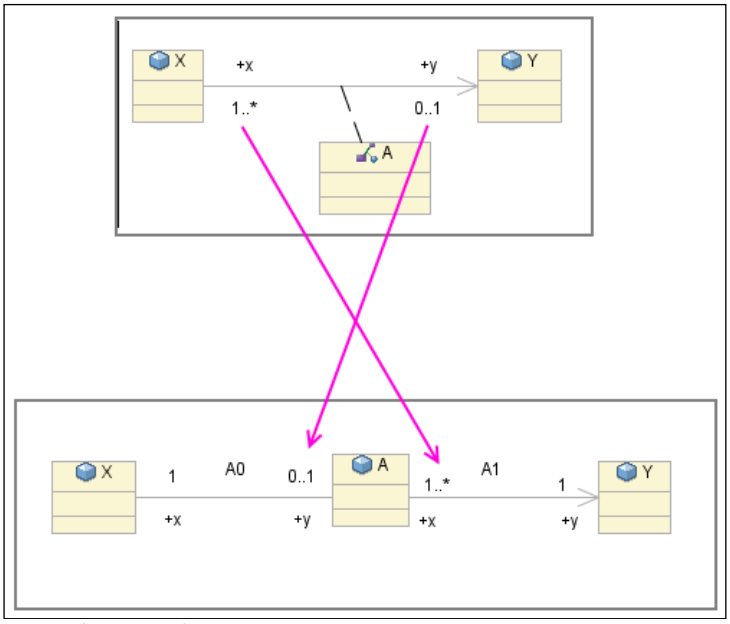

Fig. 30. Association Class replacement 'cross-over'

ACKNOWLEDGMENTS

Thank you to Simon Knight for his review of this paper.
Thank you to Christian Kohls for his shepherding of this paper.

REFERENCES

[Wolfram] Wolfram Research : Graph Theory
https://mathworld.wolfram.com/topics/GraphTheory.html

[Wolfram-Hyperedge] Wolfram Research : Graph Theory
https://mathworld.wolfram.com/Hyperedge.html
https://mathworld.wolfram.com/Hypergraph.html

[Wikipedia] Hypergraph
https://en.wikipedia.org/wiki/Hypergraph

[Fowler] Analysis Patterns – Reusable Object Models, by Martin Fowler
ISBN 0201895420
http://www.martinfowler.com/

[Fowler-Roles] Dealing with Roles, Martin Fowler fowler@acm.org
https://www.martinfowler.com/apsupp/roles.pdf

[ONF CIM] https://www.opennetworking.org/wp-content/uploads/2018/12/TR-512_v1.4_OnfCoreIm-info.zip

GLOSSARY

ACRONYM	DEFINITION
UML	Unified Modelling Language
ONF	Open Networking Foundation https://www.opennetworking.org/

Received June 2020