Population-based Structural Health Monitoring: the Network of Structures

A Thesis submitted to the University of Sheffield

for the degree of Doctor of Philosophy in the Faculty of Engineering

by

D. S. Brennan

Department of Mechanical Engineering

University of Sheffield

December 2023

Acknowledgements

First and foremost, my sincere thanks must go to my supervisor Professor Keith Worden, without whom, this work wouldn't have been possible. I would like to thank him for his guidance, help, and support throughout this project, especially for his patience explaining mechanical engineering concepts and the notion of me abiding by any form of English grammar! A special thank you must also go to my second supervisor Professor Elizabeth Cross; thank you for your support, encouragement, and keeping me sane throughout this process.

Thank you to everyone in the Dynamics Research Group. You have all welcomed me in, and it has been a true privilege to work with a group of such talented and friendly people. Especially thanks to: Chandula Wickramarachchi for her help on the initial requirements of a PBSHM framework, and Julian Gosliga for his input on Irreducible Element models and his help conducting the case study in Chapter 6. Other thanks must go to Michael Dutchman and Robin Mills for their help with the rig setup during the same case study. I'd also like to thank Andrew Bunce and Connor O'Higgins of Queen's University Belfast for the data used in Chapter 4.

Lastly, but by no means least, I would like to thank my family and friends for their support and encouragement during this work. Particularly, thanks to Sam for not only encouraging me to do a PhD, but for putting up with me over the last three months while I have been working evenings and weekends to get this finished. Morgan, sorry that you have had to miss so much time with Daddy over the last few months!

Dedicated to Morgan

Abstract

Structural Health Monitoring (SHM), at its core, is the process of monitoring a system or structure with the objective of utilising the acquired data to access the overall condition — the *health* — of the object in question. One problem of such an approach is acquiring *enough* data to specify the 'health' state of a structure. Population-based Structural Health Monitoring (PBSHM) is a recent development within the SHM community which attempts to bypass the data bottlenecks present in the 'classic' SHM scenario.

The aim of PBSHM is that by monitoring multiple structures — the *population* — one can gain additional insights into the health of a particular structure when using population data, compared to the insights available when using only a single structure's data. PBSHM operates under the premise that learnt knowledge may be shared across structures in the population; however, before any knowledge is shared, a similarity between structures is first established, to guide if an attempt to share knowledge should occur. The work conducted in this thesis is focussed on the similarity-assessment portion of PBSHM and is divided into two parts.

The first part of the thesis, focusses on developing the required language and ecosystem for determining the similarity of structures. To achieve this, a radical reconstruction of the core concepts, definitions, and language of an Irreducible Element (IE) model — the vehicle used within PBSHM to describe a structure — was required, to provide a standardised representation of structures. A novel ecosystem is then introduced to provide a shared-domain context for PBSHM similarity computations: *network*, *framework*, and *database*.

The second part of the thesis, focusses on the problems faced during the similarity

assessment process. When an IE model is curated, there is inherent author bias present within the model which introduces nuanced variations within the model. The *Canonical Form* is introduced as the potential vehicle for facilitating the *network* to adapt to these aforementioned variations. A case study is further given of generating an IE model for a real-world aircraft, a new graph notation is also introduced to enable concise pictorial representations of IE models.

Publications

Journal Papers

D. S. Brennan, T. J. Rogers, E. J. Cross, and K. Worden, 'On calculating structural similarity metrics in population-based structural health monitoring', Accepted in Data-Centric Engineering, 2024.

D. S. Brennan, J. Gosliga, E. J. Cross, and K. Worden, 'Foundations of population-based SHM, Part V: Network, framework and database', Mechanical Systems and Signal Processing, vol. 223, p. 111602, Jan. 2025.

C. T. Wickramarachchi, J. Gosliga, A. Bunce, D. S. Brennan, D. Hester, E. J. Cross, and K. Worden, 'Similarity assessment of structures for population-based structural health monitoring via graph kernels', Structural Health Monitoring, p. 14759217241265626, Aug. 2024.

D. S. Brennan, J. Gosliga, P. Gardner, R. S. Mills, and K. Worden, 'On the application of population-based structural health monitoring in aerospace engineering', Frontiers in Robotics and AI, vol. 9, p. 840058, Nov. 2022.

Book Chapters

L. A. Bull, I. Abdallah, C. Mylonas, L. D. Avendaño-Valencia, K. Tatsis, P. Gardner, T. J. Rogers, D. S. Brennan, E. J. Cross, K. Worden, A. B. Duncan, N. Dervilis, M. Girolami, and E. Chatzi, 'Data-centric monitoring of wind farms', in Data Driven

Methods for Civil Structural Health Monitoring and Resilience, 1st edition, Boca Raton: CRC Press, 2023, pp. 120–180.

Conference Papers

C. Kent, C. O'Higgins, D. Hester, D. S. Brennan, Z. Zhu, and S. Taylor, 'Expanding IE model applications with real-world case studies of bridge structures', *Submitted to International Modal Analysis Conference XLII, Orlando, Florida, USA*, 2024.

A. Bunce, D. Hester, and D. S. Brennan, 'Where is the end of a Bridge (model)?', *Proceedings of EuroDyn XII, Delft, Netherlands*, 2023.

D. S. Brennan, E. J. Cross, and K. Worden, 'A comparison of structural similarity metrics within population-based structural health monitoring', *Proceedings of the International Workshop on Structural Health Monitoring, Stanford, California, USA*, 2023.

G. Delo, C. Surace, K. Worden, and D. S. Brennan, 'On the influence of structural attributes for assessing similarity in population-based structural health monitoring', *Proceedings of the International Workshop on Structural Health Monitoring, Stanford, California, USA*, 2023.

D. S. Brennan, T. J. Rogers, E. J. Cross, and K. Worden, 'Calculating structure similarity via a graph neural network in population-based structural health monitoring: Part II', *Proceedings of the International Modal Analysis Conference XLI, Austin, Texas, USA*, 2023.

D. S. Brennan, T. J. Rogers, E. J. Cross, and K. Worden, 'On quantifying the similarity of structures via a graph neural network for population-based structural health monitoring', *Proceedings of the International Conference of Noise and Vibration Engineering, Leuven, Belgium*, 2022, p. 9.

G. Delo, A. Bunce, E. J. Cross, J. Gosliga, D. Hester, C. Surace, K. Worden, and D. S. Brennan, 'When is a bridge not an aeroplane? Part II: a population of real structures', *Proceedings of the European Workshop on Structural Health Monitoring, Palermo, Italy*, 2022, pp. 965–974.

D. S. Brennan, R. S. Mills, E. J. Cross, K. Worden, and J. Gosliga, 'On a

description of aeroplanes and aeroplane components using irreducible element models', *Proceedings of the International Modal Analysis Conference XL, Orlando, Florida, USA*, 2022, p. 14.

D. S. Brennan, J. Gosliga, E. J. Cross, and K. Worden, 'On implementing an irreducible element model schema for population-based structural health monitoring', *Proceedings of the International Workshop on Structural Health Monitoring, Stanford, California, USA*, 2021, p. 11.

D. S. Brennan, C. T. Wickramarachchi, E. J. Cross, and K. Worden, 'Implementation of an organic database structure for population-based structural health monitoring', *Proceedings of the International Modal Analysis Conference XXXIX, Bethel, Connecticut, USA*, 2021, pp. 23–41.

C. T. Wickramarachchi, D. S. Brennan, W. Lin, E. Maguire, D. Y. Harvey, E. J. Cross, and K. Worden, 'Towards population-based structural health monitoring, Part V: networks and databases', *Proceedings of the International Modal Analysis Conference XXXIX, Bethel, Connecticut, USA*, 2021, pp. 1–8.

Table of Contents

List of Figures **xiii**

List of Tables **xvii**

1 An Introduction to Population-based Structural Health Monitoring 1

1.1 Structural Health Monitoring 1

1.1.1 Damage detection 2

1.2 Population-based SHM 3

1.2.1 Irreducible element models 5

1.2.2 Graphs & networks 6

1.2.3 Similarity metrics 7

1.3 Thesis Overview 7

1.3.1 Chapter breakdown 8

2 Background 11

2.1 A brief history of Population-based SHM 11

2.1.1 The 'Towards' series 12

2.1.2 The 'Foundations' series 15

2.1.3 Summary 16
2.2 Embedding Structural Knowledge and Similarity Metrics 17
2.2.1 Knowledge as graphs 19
2.3 Shared Data Domain 20
2.3.1 SHM database 21
2.4 Conclusion 25

3 Irreducible Element Models **27**

3.1 Background 27
3.2 Root Objects 31
3.3 Free and Grounded Models 32
3.4 Ground Element 33
3.5 Regular Element 34
3.5.1 Coordinates 36
3.5.2 Context 37
3.5.3 Geometry 38
3.5.4 Material 43
3.6 Perfect Relationship 45
3.7 Connection Relationship 47
3.8 Joint Relationship 49
3.9 Boundary Relationship 51
3.10 Conclusion 52

4 Network, Framework, & Database **55**

4.1 Background 56

4.1.1 Database 57
4.2 Network, Framework, and Database 60
4.2.1 The network 61
4.2.2 The framework 62
4.2.3 The database 65
4.3 Implementation 67
4.3.1 Database choice 67
4.3.2 JSON examples 73
4.3.3 Framework 73
4.3.4 Download 75
4.4 Conclusion 76

5 Similarity Metrics **79**
5.1 Background 80
5.2 Canonical Form 85
5.2.1 Canonical form reduction rules 87
5.2.2 Jaccard index results 93
5.2.3 Reality model 95
5.3 Graph Matching Network 98
5.4 Conclusion 104

6 From Bridges to Aeroplanes **107**
6.1 Where is an Aeroplane not a Bridge? 108
6.2 The GARTEUR Structure 111
6.2.1 Attributed graph notation 113

6.3 Hawk T.Mk1 . . . 117
6.3.1 Geometrical data method . . . 119
6.3.2 Elements and relationships . . . 122
6.3.3 Graphs . . . 126
6.4 Conclusion . . . 128

7 Conclusions and Future Work **131**
7.1 Future Work . . . 136

A PBSHM schema **141**
A.1 Root Schema . . . 141
A.2 IE Model Schema . . . 142
A.2.1 Regular element . . . 143
A.2.2 Ground element . . . 155
A.2.3 Perfect & boundary relationship . . . 155
A.2.4 Connection relationship . . . 157
A.2.5 Joint relationship . . . 158
A.2.6 Shared objects . . . 160
A.3 Channel Schema . . . 163

B Real-world aircraft **167**

Bibliography **175**

List of Figures

3.1 An example of multiple AG representations of the same bridge 29

3.2 Original IE model language 30

3.3 IE model *root object* properties 31

3.4 Language migration from original to proposed IE model language . . 32

3.5 Proposed hierarchical IE model language 33

3.6 IE model [regular] *element* hierarchical properties 35

3.7 IE model [regular] *element coordinates* object hierarchical properties . 36

3.8 IE model [regular] *element contextual* object hierarchical properties . 37

3.9 IE model [regular] *element geometry* object hierarchical properties . . 39

3.10 IE model [regular] *element geometry faces* object hierarchical properties 40

3.11 An example between a standard commonly-known name geometrical representation and a complex geometrical representation 41

3.12 An example of how a *bounding box*, *faces* and complex geometrical shape interact 42

3.13 IE model [regular] *element material* object hierarchical properties . . 44

3.14 IE model [perfect] *relationship* hierarchical properties 46

3.15 IE model [connection] *relationship* hierarchical properties 47

3.16 An example of where a [connection] *relationship* could be utilised within a bridge 48

3.17 IE model [joint] *relationship* hierarchical properties 50

3.18 IE model [boundary] *relationship* hierarchical properties 52

4.1 Data transmission from structures to the PBHSM database 57

4.2 RDBMS structure example 58

4.3 NoSQL structure example 59

4.4 Visual representation of the 'network of structures' 61

4.5 PBSHM Framework hierarchical structure 63

4.6 *structure* hierarchical properties 70

4.7 *channel* hierarchical properties 71

4.8 [grounded] IE model JSON example 72

4.9 *channel* JSON example 74

4.10 PBSHM Framework IE model similarity module 75

4.11 PBSHM Framework channel data module 76

4.12 Jaccard Index embedded type comparison 77

5.1 Similarity score-driven network 80

5.2 Two-span beam-and-slab bridge example 81

5.3 Potential variations introduced into an IE model through author bias 82

5.4 Jaccard Index similarity results 84

5.5 Network diagram using the Canonical Form 86

5.6 Individual ground reduction rule 88

5.7 Perfect Joint Joint reduction rule 90

5.8 Perfect reduction rule 92

5.9 Jaccard Index similarity results using the CFRR 94

5.10 Reality model hierarchical properties 96

5.11 Maximum common subgraph diagram 98

5.12 Graph matching network similarity results 101

5.13 Jaccard Index using CFRR verses graph matching network similarity results 102

6.1 Hawk T.Mk1 fuselage 109

6.2 Hawk T.Mk1 left wing 110

6.3 GARTEUR Structure 112

6.4 Repeated vertex/edge notation 114

6.5 Repeated subgraph notation 115

6.6 GARTEUR AG representation using PBSHM notation 116

6.7 Hawk T.Mk1 fuselage, vertical stabiliser & landing gear segmentation into [regular] *element*s 118

6.8 Hawk T.Mk1 fuselage measurements 119

6.9 Hawk T.Mk1 fixed reference 120

6.10 Hawk T.Mk1 measurement equipment 121

6.11 Hawk T.Mk1 wings & horizontal stabiliser segmentation into [regular] *element*s 123

6.12 Hawk T.Mk1 landing gear subgraph 126

6.13 Hawk T.Mk1 fuselage and vertical stabiliser graph 127

6.14 Hawk T.Mk1 fuselage, wings, and horizontal stabilisers graph 128

A.1 [regular] *element* →*material* types 150

A.2 [joint] or [connection] *relationship* →nature types 159

B.1 GARTEUR AG representation without PBSHM notation 168

List of Tables

6.1 Hawk T.Mk1 fuselage and vertical stabiliser [regular] *element*s 124

6.2 Hawk T.Mk1 fuselage and vertical stabiliser *relationship*s 125

A.1 Structure object 141

A.2 Model object 142

A.3 Irreducible Element object 142

A.4 [regular] *element* object 143

A.5 [regular] *element* →*coordinates* object 143

A.6 [regular] *element* →*coordinates* →global object 144

A.7 [regular] *element* →*contextual* object 144

A.8 [regular] *element* →*geometry* object 145

A.9 [regular] *element* →*geometry* standard named types 146

A.10 [regular] *element* →*geometry* →cuboid bounding box object 147

A.11 [regular] *element* →*geometry* →faces object 147

A.12 [regular] *element* →*geometry* →face side object 148

A.13 [regular] *element* →*geometry* →face translation object 148

A.14 [regular] *element* →*geometry* complex types 149

A.15 [regular] *element* →*material* object 150

A.16 [regular] *element* →*material* →properties object 151

A.17 [regular] *element* →*material* →properties unit-free types 151

A.18 [regular] *element* →*material* →properties unit-based types and accepted values 152

A.19 [regular] *element* →*material* →conditional value object 153

A.20 [regular] *element* →*material* →conditional value →environmental object 154

A.21 [regular] *element* →*material* →conditional value types and accepted properties values 154

A.22 [ground] *element* object 155

A.23 [perfect] and [boundary] *relationship* object 155

A.24 [perfect] or [boundary] *relationship* →named element object 156

A.25 [perfect] or [boundary] *relationship* →coordinate object 156

A.26 [perfect] or [boundary] *relationship* →coordinate →global object . . . 156

A.27 [connection] *relationship* object 157

A.28 [connection] *relationship* →named element object 157

A.29 [joint] *relationship* object 158

A.30 [joint] or [connection] *relationship* →nested nature object 158

A.31 [joint] *relationship* →degrees of freedom object 159

A.32 [joint] *relationship* →degrees of freedom →global object 159

A.33 [joint] *relationship* →named element object 160

A.34 Nested type object 160

A.35 Coordinates objects 161

A.36 Value object and accepted type values 162

A.37 Bounded value object and accepted type values 162

A.38 Dimension object and accepted type values 163

A.39 Channel object 163

A.40 Channel types and accepted values 166

A.41 Channel value object 166

B.1 Hawk T.Mk1 left wing and stabilisers [regular] *element*s 169

B.2 Hawk T.Mk1 landing gear [regular] *element*s 170

B.3 Hawk T.Mk1 [ground] *element*s 170

B.4 Hawk T.Mk1 left wing and vertical stabiliser *relationship*s 171

B.5 Hawk T.Mk1 fuselage to vertical and horizontal stabilisers *relationship*s172

B.6 Hawk T.Mk1 landing gear *relationship*s 173

Chapter 1

An Introduction to Population-based Structural Health Monitoring

This chapter aims to provide a brief introduction into Population-based Structural Health Monitoring (PBSHM). Structural Health Monitoring (SHM) is first introduced, along with the potential pitfalls in the current methodology and where PBSHM aims to solve these issues. An overview of the current state of similarity within PBSHM is given, and finally a breakdown of the thesis is provided.

1.1 Structural Health Monitoring

Structural Health Monitoring (SHM) [1, 2], at its core, is the process of monitoring a system or structure with the objective of using the acquired data to assess the overall condition — the *health* — of the system in question. In short, the purpose of SHM, is to detect damage within a structure. The statement of damage detection is generally accepted to be defined as a set of hierarchical steps:

1. *Detection*: establish that damage is present.

2. *Localisation*: assess where the damage is located.

3. *Classification*: determine what type of damage has occurred.
4. *Assessment*: calculate the extent of the damage.
5. *Prediction*: estimate the lifespan.

Rytter [3] introduced the initial four hierarchical steps, Worden & Dulieu-Barton [4] later expanded this concept to include the third step: the concept of determining what type of damage has occurred. Being able to define 'damage' as definitively as outlined above — as alluded to in [4]: is a key factor in any decisions regarding if a structure is 'healthy', as any damage present may affect the designed safety thresholds and the accepted operational conditions of a structure.

1.1.1 Damage detection

How one detects damage in a structure, is another matter entirely. There are two widely-explored approaches within the community: model-based and data-based. A model-based approach would often consist of a physics-based model, commonly a Finite Element (FE) model. This model portrays a digital representation of the physical and structural properties of the structure. As data from the system monitoring becomes available, the model is often updated in an attempt to match the real-world structure as closely as possible.

This latter point highlights one of the main road blocks in a model-based SHM approach: building a model which matches exactly the dynamic behaviour of a real-world structure is incredibly difficult; any information not included within the model, provides a gap between the dynamic behaviour of the model and the real-world structure. One could simply state that this is not a problem, and instead time and care must be required when curating the model; however, the problem is that not all information may be known or even available during the development of the model.

The second approach is a data-driven approach; instead of necessitating pre-existing knowledge on a structure's composition and properties like the model-based approaches, a data-based approach assumes that no pre-existing knowledge is available on the dynamic behaviour of the structure and instead builds a statistical model of the structure based upon known 'healthy' states of the structure. As the name suggests, a data-based approach requires a certain amount of data from

a monitoring campaign — capturing data from a series of sensors attached or associated with the structure over a period of time — potentially ongoing — to build the statistical model. Raw sensor data will seldom provide a direct indication of damage presence, unless total structure failure occurs; instead, raw data are processed and combined to curate associated *feature*s, and, particularly important within the context of SHM; damage sensitive features.

A damage sensitive features is — as the name suggests — any feature derived from raw sensor data whose value would change when damage is present within a structure. A common example of such a damage-sensitive feature, would be the natural frequency of a structure. If one considers how the natural frequency is derived from the mass and stiffness, it would often prove impractical to directly measure the mass of a structure; however, if the observed natural frequency changes we can surmise that either the mass or stiffness of the structure has changed; subsequently, one could conclude that damage has potentially occurred within the system.

Akin to the model-based approach, the downfall of a data-based approach is again data availability; however, there may be a solution to the data-scarcity problem within a data-based SHM domain — Population-Based Structural Health Monitoring.

1.2 Population-based SHM

Population-based Structural Health Monitoring (PBSHM) [5–8] aims to solve the inherent data scarcity problems in a traditional data-driven SHM scenario by expanding the scope of 'structure' being monitored. The intention of PBSHM is that by monitoring multiple structures — *the population* — one can gain additional insights into the health of a given structure when using population data, compared to those available from the single structure's data.

But how does one gain additional insight on a structure's health simply by utilising monitoring data across multiple structures? By the idea that learnt knowledge can be shared. PBSHM works under the premise that knowledge gained on one structure, may then be transferred over to another structure; a process called *Transfer Learning* [9, 10]. The standard processes of traditional SHM still hold: data are captured via

a monitoring campaign, features are extracted from the raw data in the monitoring campaign, and then healthy or damaged state labels are attached to these extracted features. However, in a PBSHM approach, if for instance, one structure is missing damage-state labels, knowledge of what damage states for this structure look like could instead be learnt from another structure which has known damage states.

Transfer learning may at first appear like a panacea technology for SHM; however, like all seemingly too-good-to-be-true solutions, there is a catch. In the scenario concerning missing damage-state labels, the hope is that the shared knowledge from one structure, aids in overall knowledge of the structure in question. This may not always be the case, what happens when this transferred knowledge not only doesn't aid in the overall knowledge of the structure, but instead hinders the knowledge and potentially provides a false account towards the health of a structure? These cases are known as *negative transfer*.

PBSHM hopes to sidestep the issue of negative transfer by borrowing an idea from *domain adaptation* — a branch of transfer learning — by which knowledge can only be classed as 'shared' if both the source and target systems share a common domain. The belief in PBSHM is that if two structures share a form of similarity between themselves, it is conceivable that knowledge may be shared across the structures in a positive knowledge transfer, and the possibility of negative transfer can be reduced.

Structures are therefore categorised into two types of populations within PBSHM; *homogeneous* populations where the structures inside the population are considered to be nominally identical, and *heterogeneous* populations where the structures are considered diverse. An example of a homogeneous population would be of a wind farm: generally wind turbines within a farm are of the same manufacturer and type and can be classed as uniform in nature; the only variations present between structures are manufacturing tolerances. In contrast, a heterogeneous population would contain structures with marked differences, for instance aeroplanes, bridges, and wind turbines.

Whilst the similarity between homogeneous structures is evident, the similarity between heterogeneous structures is not so apparent. How can two structures be similar when — by their very nature as being heterogeneous — they are diverged? This is where PBSHM employs the premise of a *substructure*. Both of the structures being compared may be heterogeneous in nature, one a wind turbine and the other an aeroplane; however, there may be a smaller common pattern within both structures.

This substructure could vary from a single shared component, to a complex pattern or purpose such as an aerofoil.

To achieve the goals of PBSHM therefore, two basic problems require addressing: finding a mechanism for computing the degree of similarity between structures (*or components of the structure*), and subsequently, transferring learnt knowledge across the established population. The body of work contained within this thesis, focusses on the first problem of PBSHM: computing the degree of similarity between structures.

1.2.1 Irreducible element models

Before one can determine the similarity of two structures, there must first be a shared context in which both structures are described. If within one structure a component is referred to as a 'bolt' and in the other structure it is referred to as a 'cylindrical bar', similarities between the two structures are going to be challenging to compute. What is required, is a shared methodology and language to facilitate deconstructing a structure into known components. This shared language for describing structures within PBSHM is *Irreducible Element* (IE) models [6].

The premise of an IE model is to break down a structure into the *structurally-significant* components which make up a structure. Imagine hitting a structure with a magic hammer which instantly removed any binding items — bolts, welds, rivets, etc. — you would be left with a pile of parts that comprise the makeup of the structure. This — in a very simplified manner — is the idea behind an IE model. Any available knowledge on each component is embedded into the model: the form, material, and composition. Interactions between these components are also included within the model with any relevant knowledge on how the components interact.

The belief of PBSHM, is that via the description of these structurally-significant components — and the interactions between these components — a model can be arrived upon which encapsulate the very essence of the structure in a manner which enables comparisons to be drawn between the structures.

The keen-eyed reader will have noticed the prefix of 'structurally-significant' against the definition of structure components. Whilst there are no restrictions within the language of an IE model as to what may be considered a component, one does have

to ensure that any component included within a model aligns with the spirit in which the 'rules' of an IE model were created. It could be foreseen that a modeller may desire to include every 'nut and bolt' within the structure; however, one has to ask; is the bolt itself a significant structural component within the model, or is the bolt simply part of the nature as to how two components are held together in static motion. One has to keep the context as to why these models are being created, whilst the inclusion of minute details such as every 'nut and bolt' may appear as an effort of completeness, the modeller may in fact be hindering the very purpose as to why the model exists: comparing the similarities between two structures.

1.2.2 Graphs & networks

Graphs are commonly used to represent interlinked data as a whole object. They enable complex structural relationships to not only be represented mathematically, but to facilitate a map of what is connected to where [11, 12]. A data point within a graph is a *vertex* and the connection between two data points is an *edge*.

If one considers the required relations to be understood within the context of determining similarity within PBSHM, having a method to represent these complex interlinked data points as a whole object is incredibly useful. Firstly, to understand the relationships between the components within a single structure; secondly, to understand the computed relationships of similarity scores between each structure.

*Graph*s are a representation of data within the field of *Graph Theory*, *Network*s are a representation of data within the field of *Network Science*. Both graphs and networks are constructed in the same manner and model the same interlinked data — graphs (vertices & edges), networks (nodes & links) — and as such, they are often used interchangeably; however, the associated connotations of using a certain terminology can differ depending upon the research field. In an effort to provide a standardisation within PBSHM — or at least within this thesis — the terms will be used when referring to specific scenarios. When referring to an individual representation of a structure and the associated relationships within the structure, this will be called a *graph*. When referring to a group of structures and the associated relationships between structures, the term *network* will be used.

All edges within PBSHM graphs and networks are undirected edges: the relationship between the two data points (V_A, V_B) is universal and is valid if travelling from $V_A \rightarrow$

V_B or $V_B \rightarrow V_A$; subsequently, the graphs — and networks — can be considered as undirected graphs. A graph within PBSHM may not have any *isolated* vertices — a vertex without any edges. A network within PBSHM may be considered a *connected* network — a network where every node has a link to every other node — and the links *weighted* — where each link is weighted with a value. The network within PBSHM may also be considered a *multinetwork*: where each node pair may have multiple links connecting them.

1.2.3 Similarity metrics

The final destination for a structure — within the similarity concept — in PBSHM is determining the similarity between itself and other structures. It is envisioned that within the field of PBSHM there will eventually be multiple different methods and algorithmic implementations to compute a similarity value between two IE models; however, the output from the algorithms must be standardised to facilitate equal comparison between the structures.

These computed values are what is referred to within PBSHM as the similarity metrics or similarity matrix. Each algorithm will — regardless of the implementation — ultimately provide a value between 0 and 1 when provided with two IE models, where 0 implies that the IE models for the underlying structures have no similarities and 1 implies that the IE models are considered identical for the associated structures. These values then live within the network of structures defined within this thesis.

1.3 Thesis Overview

The body of work within this thesis focusses on each step within the similarity lifecycle within PBSHM and forms the necessary technical and fundamental building blocks required to facilitate a *network of similarity* in PBSHM.

The first half of this thesis is dedicated to providing a holistic computational ecosystem for PBSHM to further the desires of transferring knowledge between established populations of similar structures. This holistic ecosystem required a radical reformulation of the very concept of an IE model, to support an enhanced

structural knowledge embedding and provide a standardised language for describing structures to facilitate the unhindered similarity network. The aforementioned ecosystem required the conceptualisation and implementations of three shared data domains: data residency, computational operations, and similarity comparisons.

The second half of this thesis utilises the introduced ecosystem and reformulation of IE models and explores its use within real-world scenarios. Firstly, an example of inherent model variations from author bias is explored and how the reformulated concept of IE models now contains knowledge regarding these biases. Using the newly embedded knowledge enables these variations to be reduced within the shared comparison domain by the introduction of the Canonical Form. Secondly, a machine learning method for computing the similarity within the shared comparison domain is utilised. Lastly, the introduced language of IE models is applied to a real-world aeroplane and a new graph notation is introduced to aid in the visualisation of IE models.

1.3.1 Chapter breakdown

- Chapter 2 reviews the current work published in PBSHM and explores potential database systems and knowledge embedding available for SHM.
- Chapter 3 expands upon the previous work in IE models by introducing a radical reformulation of the IE model concept to facilitate increased structural knowledge embedding within the model. The new language of IE models provides a standardised description of structures within PBSHM.
- Chapter 4 introduces a holistic computational conceptualisation for PBSHM via the definitions of a PBSHM ecosystem: database, framework, and network. This chapter also provides an initial implementation of the technical components of the ecosystem: database and framework.
- Chapter 5 evaluates the existing similarity metrics used within PBSHM against the inherent human variations introduced into an IE model by author bias. With the increased knowledge now embedded within an IE model, the author bias can be reduced by the introduction of a Canonical Form; a reduced IE model which provides a standard representation for a single structure for use within the network. Finally, a Graph Matching Network is investigated as a potential similarity metric within PBSHM.

- Chapter 6 demonstrates the introduced IE model language against a real-world aeroplane. New graph notation is introduced for visualising IE models.
- Chapter 7 concludes this thesis and discusses potential future work.

Chapter 2

Background

While the focus within this thesis is on the similarity assesment aspects of PBSHM, it is important to understand the wider context of PBSHM and the current state of literature within the field. Within the scope of PBSHM, there is relatively little literature on the subject of similarity — aside from the initial papers introducing the notion; however, the problems faced within the similarity aspect of PBSHM have been explored in other fields.

As such, this chapter is broken down into two parts, the first part in Section 2.1 covers a brief history of PBSHM to date, in the aim of providing a background to the world in which this thesis is based. The second portion of this chapter is dedicated to exploring the surrounding literature on the main problems to which this thesis aims to provide a solution: the problem of embedding structural knowledge and generating similarity metrics is explored in Section 2.2 and the problem of a shared-data domain is explored in Section 2.3.

2.1 A brief history of Population-based SHM

Population-based Structural Health Monitoring (PBSHM) is a relatively-recent development within the field of Structural Health Monitoring (SHM); as such, the available literature on the subject is minimal — compared to the available literature on SHM — and has been mainly conducted over a five-year period prior to the date of this thesis. The first mentions of a population-based approach to SHM were

made in 2015 [13–15].

Papatheou *et al.* [13] first mention the notion of a population when predicting the power curve of a turbine within a wind farm. Instead of treating each turbine as a siloed structure, they instead treat the wind farm as a homogeneous population of structures and are able to predict the power curve of one turbine by utilising data from other turbines within the farm. Antoniadou *et al.* [14] also remark upon the possibility of a population-based approach in their quest for detecting damage in turbine gearboxes and blades within the same wind-farm setting. Worden *et al.* [15] further solidifies the population premise, by defining the scope of PBSHM as developing a methodology to take inferences learnt from one structure and apply these learnt inferences to other structures.

After the initial suggestion of a population-based approach to SHM was mentioned in [13–15], two further papers introduced the first technical definitions within PBSHM. Bull *et al.* [16] introduced the concept of a population *form* and Gosliga *et al.* [17] introduced the concept of an Irreducible Element (IE) model and the associated Attributed Graph (AG) representation. The form is a model — in a defined feature domain — which is considered a generic representation of the population. An IE model represents the physical composition of an individual structure within a population. These later papers by Bull *et al.* and Gosliga *et al.* were followed up by a series of papers expanding on these initial theorems and setting the potential scope for PBSHM: the 'Towards population-based structural health monitoring' series [18–23] (henceforth referenced as the 'Towards' series).

2.1.1 The 'Towards' series

In the first part of the 'Towards' series, Bull *et al.* formalised the understanding of the population form [18] as a model that captures the very essence of the structures within a *homogeneous* population. Bull *et al.* also introduces the first formalised definition for the two types of populations within PBSHM using graph theory [11, 12]; a *homogeneous* population, and a *heterogeneous* population. A population can be classed as homogeneous, if the associated graphs of the member structures can be determined to be pair-wise topologically equivalent with structural property values (graph attributes), considered to be from the same base-distribution. If any structure within the population is determined to be non-homogeneous, then the

population is defined as a heterogeneous population. Bull *et al.* also introduce the special case of a *strongly-homogeneous* population, where structures are determined to be identical — usually because of them being the same make or model — with any variations considered to be present resulting from the associated manufacturing or embodiment process.

In the second part of the 'Towards' series, Gosliga *et al.* [19] expand upon the proposed IE model description of structures. Different areas of knowledge are discussed and evaluated for the inclusion within an IE model as well as introducing the reasoning behind the desire for an IE model to subsequently be converted into an AG. Graphs provide a potential platform for enabling the quantification of similarities between structures. Determining if two structures are similar or not, is a key component in PBSHM to enable the decision as to if a population is homogeneous or heterogeneous, as well as providing an insight into if transfer learning is possible or not (introduced in Part IV [21]).

Gosliga *et al.* [20] also introduce in the third part of the series, the initial vehicle for determining the similarity between two structures: the Jaccard Index or Jaccard similarity coefficient [24, 25]. The Jaccard Index — in generic terms — is a method for generating a similarity metric between two datasets. In the context of a graph domain, the Jaccard Index can be used to generate a metric of similarity between two graphs using the output from a Maximum Common Subgraph (MCS) algorithm [26]. The MCS — or Maximum Common Induced Subgraph [27] — is the process of finding the largest common subgraph that is present in two graphs G_1 and G_2. The chosen implementation of the MCS algorithm within the paper is the Bron-Kerbosch clique-finding algorithm [28]. Gosliga *et al.* also mention the notion by which similarity metrics within a population relate to each other: the *network of structures*.

The fourth paper in the 'Towards' series [21] by Gardner *et al* addresses the problem of transferring learnt knowledge between structures. Gardner *et al.* introduce the field of *Transfer Learning* [9, 10] as the aforementioned vehicle to accomplishing the knowledge transfer required for PBSHM to function. Machine Learning (ML) [29] is the process of teaching a computer to learn a particular piece of knowledge from existing data, in the ambition that when new and unseen data are presented to the computer, it can interpret and make a desired decision. This decision may often be in the form of a classification; in the case of SHM, this could be as rudimentary as determining if damage is present or not within a structure.

Traditionally ML has often limited its algorithms to having the learning and future data be from the same domain. Transfer learning is a subfield within ML where the objective is that the learning data and the future data do not have to be from the same domain. This new field unfortunately leads to another problem, negative transfer — where the transfer of learnt knowledge from the source domain to the target domain, negatively impacts the knowledge in the target domain. Gardner *et al.* acknowledge the possibility of negative transfer within the context of PBSHM and reinforce the idea that the potential solution for limiting its effect, is to establish a similarity between structures before facilitating any knowledge transfer. Therefore, Gardner *et al.* highlight the importance of the IE model and similarity metric work introduced in [19, 20] to establish this necessary similarity in the populations.

It is important to note that this is not the first time that ML has been applied to an SHM scenario. Farrar and Worden [1] outline the key approaches for applying statistical pattern recognition and ML techniques to the problem of SHM. Worden and Manson [30] apply these highlighted ML techniques to a number of data-driven SHM case studies. This may appear somewhat parochial, given that the authors of the aforementioned papers are from the same research group as the author; however, the intention here is to simply highlight that ML approaches have been applied to SHM before and not to provide an exhaustive list.

Part Six in the 'Towards' series by Worden [22] explores the use of a mathematical space in which the transfer of knowledge can occur. Worden hypothesises that the transfer of knowledge will occur as a set of mappings and associations between the feature spaces for the associated SHM-driven problems. The population therefore, would have a shared feature space which would be the union of all the member structures' feature spaces. The proposed field for transferring knowledge within this shared feature space is that of *fibre bundles* [31].

The final paper in the 'Towards' series (the seventh) is by Lin *et al.* [23] and is focussed on the presence of Environmental and Operational Variations (EOV's) within PBSHM. Data acquired from lab experiments is often shielded from the inherent fluctuations of environmental and operational conditions that are common in real-world SHM data. In a lab, the temperature will often be fixed during an experiment; unfortunately this is not possible when measuring data on an operational structure; as such, the effect that EOV's have on operational structure data, must be managed within any SHM scenario. Lin *et al.* explores how the use of a population-based approach provides the required data to create a population

map in a strongly-homogeneous population; a model which can encapsulate the effect that EOV's have upon the population and can identify when the EOV effect is unexpected.

2.1.2 The 'Foundations' series

The 'Towards' series set the scope and initial premise for a population-based approach towards SHM; however, this initial series of scoping papers required a follow-up series to formally cement the introduced concepts and summarise with examples the new terminology and methods which were to become the very foundations of PBSHM. And so the 'Foundations of population-based structural health monitoring' journal series (henceforth referenced as the 'Foundations' series) was birthed.

Part I of the 'Foundations' series by Bull *et al.* [5] provides the definition of a *homogeneous* population — and thereby the definition of a *heterogeneous* population — and the case for when a homogeneous population can be classified as *strongly-homogeneous*. The definition of a population *form* capturing the very essence of the structures within the homogeneous population is also given.

Part II of the 'Foundations' series by Gosliga *et al.* [6] provides the definition of an IE model and the associated knowledge necessary to capture the dynamic nature of a structure, namely the topology, geometry, material, and boundary conditions. The paper also provides the terminology and definitions required to convert an IE model to an AG. An initial similarity metric is provided using the Bron-Kerbosch algorithm and the Jaccard Index to generate a Jaccard distance metric against a set of toy example structures.

Part III of the 'Foundations' series by Gardner *et al.* [7] focusses on the knowledge transfer strand of PBSHM. Gardner *et al.* include the definitions required for performing transfer learning — specifically by domain adaptation — on a heterogeneous population, and provide the context in which the feature space of one structure can be mapped onto another structure's feature space. The problem of negative transfer is addressed, and the similarity comparisons provided in Part II [6] are suggested as a possible method for limiting the effect of negative transfer within PBSHM.

Part IV of the 'Foundations' series by Tsialiamanis *et al.* [8] expands upon the premise of a fibre bundle to encapsulate the shared feature space associated with a population. Tsialiamanis *et al.* use a Graph Neural Network (GNN) to solve the problem of determining the normal condition features across a whole population. GNN's are a field within ML enabling learning on graph structures. Because the structures within PBSHM have a graph-domain representation — via an IE model — the GNN is a natural fit for evaluating the features across a population.

2.1.3 Summary

The overall premise of PBSHM can be summarised as: *For a given population of structures, determine the similarity between the individual member structures and subsequently, transfer any learnt knowledge across member structures where the aforementioned similarity has been established.* While the previous statement may appear simple in theory, in practise there is a whole realm of associated problems and bottlenecks that need solving in order for the methodology of a population-based approach to achieve realisation.

The previous sections of this chapter aimed to give the reader an initial overview of the history of PBSHM as well as highlighting key literature milestones in the journey towards the foundations of PBSHM. The literature included is by no means an exhaustive inspection of all published PBSHM literature to date, instead it is hoped that the included literature provides an understanding of the aforementioned problems and potential theorems and strands currently being researched into, to solve these problems. The remaining content in this chapter will focus on the literature areas required for addressing the challenges faced to facilitate the network of structures premise within PBSHM, namely: embedding structural knowledge, generating similarity metrics, and providing a shared data domain. For the interested reader, Worden *et al.* provide an extensive look at the progress to date through a summary paper [32].

The astute reader will have noticed that 'Part V' is missing from both the 'Towards' and 'Foundations' series of papers. This is because the body of work covering 'Part V' is conducted as part of this thesis, specifically the Irreducible Element model (see Chapter 3) and Network, Framework and Database (see Chapter 4) chapters. The author must also recognise that the main body of research conducted within

field of PBSHM, has — in the main part — been driven by colleagues within the same research group as the author. This by itself, is not unusual when a new field of research is developed; however, the author is aware of the potential for an 'echo chamber' effect within the initial referenced research and has attempted to maintain an open view throughout this thesis.

2.2 Embedding Structural Knowledge and Similarity Metrics

In the aid of chapter narrative, both the 'Embedding Structural Knowledge' and 'Similarity Metrics' literature is discussed as a whole, as both challenges — within the context of PBSHM — are codependent upon each other. Within the methodology of PBSHM, knowledge of a structure is embedded into an Irreducible Element (IE) model which is later converted into an Attributed Graph (AG) space to enable similarity comparisons to be drawn. Since the initial conceptualisation of IE models and the first similarity metric by Gosliga *et al.* [6, 17, 19, 20], additional material on the subject has been published.

Gosliga *et al.* [33] revisits the network of structures premise introduced in 'Towards' Part III [7] by exploring the different compositions of the network. In the proposed network structure, each connection between a structure in the network should only be formed when a given criterion is reached. This would naturally resolve to be the learnt similarity between the aforementioned structures; this counteracts the accepted standard operation of a network. Networks are often induced with a predefined set of links. In the proposed network of structures, not only do the links need to be optional, but the network needs to expand and be dynamic as new structures are included within the network. Gosliga *et al.* explore the options of the network being a fully connected or communities approach.

'Communities' in network science — like in human society — is a term for grouping sets of nodes together, where each node in the community has more in common with their community nodes, than nodes of another community. In short, it is network science term for clustering [34] like-minded nodes together. A fully-connected network is the term used to denote that every node within the network is connected to every other node in the network via a link.

With a community approach, any included structure to the network, would necessitate a similarity to be computed between the inserted structure and a representative structure for each community. In a fully-connected network, the introduced structure would necessitate comparisons to be computed between the inserted structure and every other structure in the network. The computational advantage to a community approach is clear; however, the assignment between communities is not consistent. The assignment of a community may be different depending upon the order in which the structures are added to the network and which community members are selected to represent the community during assignment.

Worden *et al.* [35] discuss the theory of how structures within PBSHM will be determined as similar — or dissimilar — and introduce the notion of a threshold of similarity which must be met before knowledge transfer between structures may occur with a reduced likelihood of negative transfer. The first example given by Worden *et al.* are of an aeroplane and a four span bridge, from a purely topological point of in the attributed graph space, they are identical apart from a single node. Worden *et al.* state that the same could be stated for a three and four-blade wind turbine.

Engineering judgement would suggest that the two wind turbines should be evaluated as having a higher similarity than the simplified aeroplane and bridge within the comparison space. Worden *et al.* highlight a potential solution to the comparison conundrum, by suggesting a tiered knowledge embedding within the comparison space to facilitate separation of structures where categories of knowledge are disparate. The tiers suggested by Worden *et al.* are topology, structure, geometrical knowledge in an IE, topological knowledge embedded in an IE, and material knowledge embedded in an IE.

Both sets of structures — the pair of wind turbines and the set including a bridge and an aeroplane — would initially appear as similar when only performing a comparison in the topological space. When the structure information is also included, the pair of wind turbines would still appear as the same level of similarity; however, this is where the bridge and aeroplane would start to move apart within the comparison space. As more knowledge is included within the comparison metric, the sets of structures would become distant within the comparison space to the point where the sets would be identifiable as similar and not similar.

Bunce *et al.* [36] introduce the first set of guidelines and rules for generating IE models of bridges. The introduced rules and methodology for breaking down a bridge into element sets for an IE model are in the effort of ensuring that structures of the same type should be able to have some form of similarity within the comparison space.

Hester *et al.* [37] produces the first paper where the rules discussed by Bunce *et al.* [36] are used to break down a real life bridge: the Bosphorus suspension bridge. The bridge underwent maintenance between 2004 till 2015 to replace and redesign the hangers used on the bridge because of fracturing found in the web plates. Hester *et al.* explore the effects that these changes had upon the Jaccard Index similarity comparison by producing an IE model of the bridge before and post hanger changes.

Gosliga *et al.* [38] expands upon the introduced bridge-specific PBSHM work outlined by Bunce *et al.* [36] and Hester *et al.* [37] by implemented IE models for five types of bridges: beam-and-slab, truss, arch, cable-stayed, and suspension. Each type of bridge has both shared and unique components to their construction technique and therefore provides a unique opportunity to evaluate the performance of knowledge embedding within PBSHM and the comparison space. The Jaccard Index is once again used to facilitate the comparisons between eight bridges across the aforementioned types.

Whilst not directly relevant to the work outlined in this thesis, it is important to note that Gosliga *et al.* also explore the use of *hypergraph*s for the first time within the context of PBSHM. The main difference between a hypergraph and graph is that a *hyperedge* — the hypergraph equivalent of an edge within a graph — can connect to multiple vertices; compared to an edge which can only connect to two vertices.

2.2.1 Knowledge as graphs

As pointed out by Gosliga *et al.* [6], PBSHM is not the first field to use graph theory to determine the similarity between two complex relational objects. Duesbury *et al.* [26] highlight the field of chemoinformatics as one of those fields which use graphs for embedding the complex composition of chemical compounds and also utilise variations of the maximum common subgraph approach to find similarity.

Raymond and Willet [39] provide a review of recent maximum common subgraph

algorithms and discuss the potential for reducing a graphical structure before comparison within any algorithm. Fooshee *et al.* [40] study the use of the maximum common subgraph method to determine the atoms that are present in both the reactants and products graphs from a chemical reaction. Cao *et al.* [41] implement a new backtracking algorithm within the context of a maximum common subgraph search for drug-like compounds within a chemical database. Ehrlich and Rarey [42] review the use of maximum common subgraphs and graph theory in the field of molecular science via chemoinformatics databases. Schietgat *et al.* [43] focus on the use of the maximum common subgraph approach in outerplanar graphs derived from molecular structures.

Chemoinformatics is by no means the only field of research where the similarity between two objects is determined by the use of graph theory. The maximum common subgraph can be found in computer malware detection [44, 45], image and video search [46, 47], and pattern recognition [48, 49], to name but three areas.

2.3 Shared Data Domain

As alluded to in the first four parts of the 'Foundations' series [5–8], there is a data requirement for the proposed framework to operate successfully. To ensure that PBSHM research is focussed on the core technology to achieve adoption, a shared-data domain is proposed, to house PBSHM data in a shared repository and format: a *database*. Databases, at their core, are simply a technique for storing data in a structured manner. Codd [50] introduced the idea of a relational database in 1970 and it has since become the foundation of modern *Relational Database Management Systems* (RDBMS).

As time has moved on, so has the desire of what a database should store. One of the key principles within a relational database is the idea of a *schema* [51, 52]: think of this as akin to an architect's drawing for a building. The schema lays out where each item of data belongs, what data are allowed, and what relationships between data should be present — a blueprint for data residency within a relational database. In theory, this database principle provides an oversight of how the data within are related and ensures only correct and complete data are included within the system; however, in a modern age where one can gain knowledge by a variety of avenues — partially complete data, number of different sources, no structure at all

— relational databases and their requirements for pre-existing knowledge in the form of a schema, fall short of meeting potential current demands and have necessitated the introduction of a new breed of databases.

Not only Structured Query Languages (NoSQLs) [53] are a family of database methodologies that aim to solve the aforementioned problems seen in relational databases, by enabling schema-free data storage. *Structured Query Language* (SQL) [54] is used to manage data inside an RDBMS, and as such, NoSQL databases are designed without these inherent language constraints. It is important to note that despite the technological hype-train [55] regarding new technology, the introduction of NoSQL databases does not mean that the relational database model should be considered obsolete and irrelevant for today's workloads. Both relational databases and NoSQL databases are tools to accomplish a job; as such, they each have their own purpose under the correct circumstances.

A concern within any technology stack is how to scale the system; in the context of a database, this can be broken down into *vertical* and *horizontal* scaling. *Vertical* and *horizontal* scaling are the principles of how to facilitate the required growth of a system; the *vertical* mode views scaling from a single item/node, the *horizontal* mode views scaling from multiple items/nodes. Nance *et al.* [56] and Zafar *et al.* [57] review these concerns in conjunction with detailing differences between RDBMS and NoSQL. Gandini *et al.* [58] outline some performance-planning considerations when using a NoSQL database. It should be noted that some security concerns highlighted by Nance *et al.* [56], regarding encryption at rest[1], authentication in a *sharded*[2] environment, and client communication encryption have been addressed in MongoDB [59–62] since the publication of these papers.

The rest of this section will focus on the literature available on the use of databases within an SHM-focussed context.

2.3.1 SHM database

Perhaps the most complete work that can serve as a direct comparison to the technological steps required within PBSHM, is the work of El-Mehalawi and Allen Miller [63, 64] in their two part series looking at the premise of a component

[1] Encrypted Storage Engines are available as of MongoDB 3.2 Enterprise [59, 60]
[2] *Sharding* is a method of splitting data across multiple database instances

database whose purpose is to provide cost estimations of new components in net-shape manufacturing.

Over the two papers, the proposed database is broken down into five technical operations: creating a computer-aided design (CAD) model in the standard for the exchange of product information (STEP) file format, converting the CAD model into an attributed graph, indexing the graphs based upon defined attributes, retrieving similar graphs when a new component is evaluated within the system, and finally producing a similarity factor between the retrieved graphs. The steps of the proposed system by El-Mehalawi and Allen Miller overlap the three key areas being discussed within this chapter; however, in the interest of maintaining a narrative, the papers are discussed only within this section of the chapter.

The first part in the series [63] looks at embedding component knowledge into an attributed graph. Akin to PBSHM, the work by El-Mehalawi and Allen Miller desires a geometrical and topological representations of a physical item to be embedded in graph form; however, rather than a graph representing a whole structure as is stated in PBSHM, the work by El-Mehalawi and Allen Miller has a graph representing only a single component. The premise is that for each component, every surface within the component becomes a vertex in the graph and every edge between the faces becomes an edge in the graph. If one were to take a shell-like cylinder, it would be stated that there were four surfaces, one annular surface at either end of the cylinder, and two cylindrical surfaces around the cylinder. Each annular surface at the end of the cylinder would have two circular edges to the corresponding cylindrical surfaces, and the two cylindrical surfaces would have two straight edges connecting to each other. This embedding of knowledge is accomplished by saving a CAD model in the STEP file format, and then further converting the STEP file format into the described attributed graph.

The second part in the series [64], explores the forming of a similarity factor upon the inclusion of a new component within the system. When a component is included within the database, El-Mehalawi and Allen Miller state that not only is the graph of the component stored, but associated metrics and statistics are included. These metrics — number of nodes, number of edges, etc — create the base for the initial retrieval of similar components from the database. Upon a new component being evaluated for similarity, the system first calculates the same metrics as stored within the database, based upon a tiered system; the search brings back graphs ordered by the similarity between the metrics of the new component and the existing component

database. The aim behind this method of retrieval is to eliminate components where there is a big difference in the graph size — number of nodes, number of edges — and performing an exact comparison on these components would be a waste of computational resources. Once only initially-similar components are left, graph comparisons can be computed against the returned set and the new component.

Jeong *et al.* [65] explore the requirements of data-management tools for SHM, particularly in the use case of sensor data from bridges. Within the building and construction industry, the building information model (BIM) [66] has emerged as an open standard to encapsulate a digital representation of physical characteristics and to provide support for sharing of building-related information. The same ethos has now been applied to bridges with the bridge information model (BrIM) in the desire to create a centralised format for capturing data associated with a bridge's lifecycle. Like most systems that need to regulate and control where data resides, BrIM uses a schema via the eXtensible Markup Language (XML) file format. The work by Jeong *et al.* investigates the implementation of a BrIM database to facilitate the storage of BrIM data in a centralised system. They use a combination of NoSQL databases — MongoDB and Cassandra — to achieve both a local and centralised version of a BrIM database.

De Oliveria *et al.* [67] design and implement a full SHM data-management system with the focus given to facilitating a low-cost hardware implementation with support for remote monitoring. De Oliveria *et al.* approach the problem using the Internet of Things (IoT) [68] paradigm: everyday *things* that can communicate with each other to achieve a common goal. They use an everyday Single Board Computer (SBC) [69] in the form of a Raspberry Pi to not only capture the data from the attached sensors, but to also store the data within a NoSQL database — MongoDB to be precise — and also serve a web management interface for monitoring and viewing the data captured remotely.

Koo *et al.* [70] build an SHM data-management system based upon an RDBMS database, a MATLAB integration, and a web interface. The principle behind the system by Koo *et al.* is that instead of data being sent directly to a centralised system, there is instead a site database running on a computer which captures sensor information from the structure. This data is then concatenated up to the central system where the data may be interrogated by a MATLAB interface for any research requirements. The data within the central system is also available via a web interface to view the available channels and to view historical time series data. Akin

to the work by El-Mehalawi and Allen Miller, Koo *et al.* store within the database a set of metrics on each dataset — generated by the MATLAB interface — to provide an overall summary for users.

Valinejadshoubi *et al.* [71] depict a case study for including SHM data into a BIM model. Valinejadshoubi *et al.* highlight the issue of the BIM specification not including any native area for SHM raw data inclusion within the model; instead, Valinejadshoubi *et al.* build a separate database via an RDBMS to store the aforementioned SHM data — strain data in this case. When SHM data are written to the database, the BIM model gets updated with the latest real-time sensor data. Every object within a BIM model must be from an Industry Foundation Classes (IFC) class, in the case of sensors this is 'IfcSensor' and 'IfcSensorType'; however, Valinejadshoubi *et al.* highlight that for wider SHM adoption within BIM, there would need to be an expansion of the IFC standard related to BIM to include additional sensor information.

Rio *et al.* [72] explore the currently allowed IFC against a set of sensor types and propose a possible extension to the IFC to natively support a wider range of sensors in a BIM model. O'Shea and Murphy [73] demonstrate that even with the current limitations in the IFC specification, the sensor network of an SHM campaign can be included within BIM using both the current class names and custom names introduced by O'Shea and Murphy.

Mita *et al.* [74] discuss another SHM database; however, Mita *et al.* propose that instead of sensors communicating with a digital acquisition system (DAC) and then the data being transmitted to a central database, the sensors are instead smart and communicate directly with a sensor gateway which writes the data into the database. Pregnolato [75] suggests the need for a national UK bridge database to enable risk-based bridge safety determinations. Testoni *et al.* [76] design a lightweight SHM sensor network based upon using data-over power for communication between the sensor nodes and the gateway machine, with the intention that the data captured can be readily stored within an SHM data-management system. McVay *et al.* [77] discuss the implementation of a state-wide bridge substructure database that acts as an information store for a bridge throughout its entire life. Arliansyah *et al.* [78] discuss the implementation of a road inventory database to track the location and reported condition of bridges within a city.

2.4 Conclusion

In conclusion, this chapter has provided a brief history of the published literature on PBSHM to date. This chapter is by no means intended to be an exhaustive survey of the literature and is instead to provide a general overview of the field and the problems faced towards the adoption of the technology.

This chapter has also explored the published literature in the problem areas covered within this thesis. It is important to note, that within the chapter there is no reference to any commercial solutions. The author is aware of the existence of such systems; however, because of the sensitive commercial nature of the solutions, it is difficult to extract any reference material for inclusion within the literature survey.

It is clear to see, that there is no existing solutions which would be a direct replacement for the methodology proposed within PBSHM. This factor dictates the clear development of the Irreducible Element (IE) model premise as the vehicle for providing a uniform description of structures and the need for a shared-data domain in which PBSHM data and computations can reside.

Chapter 3

Irreducible Element Models

The main theme running through this thesis relates to the idea of being able to generate a similarity metric between two structures; however, before this can occur, there needs to be a fair and equal representation of each structure within PBSHM, thus the purpose of the Irreducible Element (IE) model in PBSHM. This chapter radically reconstructs the core concepts, definitions, and language of an IE model, to facilitate a standardised representation of structures, regardless of the class of structure, whilst enabling additional engineering knowledge and design choices to be embedded into the model, to aid in generation of accurate similarity metrics.

This chapter firstly explains the background of IE models as well as the potential problems with the current version of IE models in Section 3.1, introduces the idea of a *root object* and the *free and grounded* models in Sections 3.2 and 3.3 respectively, introduces the supported *element* variations in Sections 3.4 and 3.5 and finally introduces the supported *relationship* variations in Sections 3.6, 3.7, 3.8, and 3.9.

3.1 Background

The notion of an Irreducible Element (IE) model was introduced by Gosliga *et al.* [6, 17, 19] as the vehicle used within PBSHM to describe structurally significant components of the system and the associated interactions between said components. This model comprises the abstract representation required to evaluate the similarity of heterogeneous structures within the shared-data domain: the *network* (see Section

4.2.1). An IE model is only concerned about components classified as belonging to the structure, as such, the model notes where interactions with external systems are present; however, it does not include any information to the extent of the effects such external systems have upon the current model.

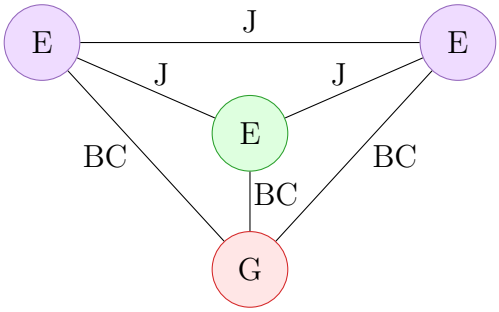
IE models by their very definition are designed to be abstract, it would be both impractical and illogical for an IE model to contain such rich 3D information as to replicate the data available within a Finite Element (FE) or Computer-Aided Design (CAD) model. Such detailed information as the complex geometrical mesh of a component is redundant when attempting to compare the overall similarity of geometry across all structures within the network. An IE model should only contain such pertinent structural information as to facilitate capturing the very essence and purpose of a structure.

An IE model describes the structure in question; however, an IE model cannot be directly inserted into the network of structures as the network exists in a graph space. Subsequently, an IE model representation is transformed into an Attributed Graph (AG), via each component within the model becoming a vertex within the graph and interactions between components becoming edges. Any properties belonging to either components or associated interactions are embedded within the graph as attributes on the respective vertex/edge.

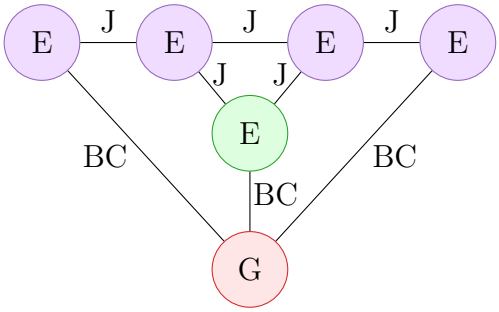
Gosliga *et al.* [38] explored the aforementioned theory with the comparison of eight bridges of different structure types. They generated an IE model for each bridge, converted these into their associated AG representation and then used Maximum Common Subgraph (MCS) with Jaccard similarity score to compare the geometrical similarities within their population. Whilst this initial work demonstrated conceptually the use of IE models within the realm of PBSHM, further expansion of the IE model language and concepts are required to support the use case of a *network of structures* within the PBSHM database.

The objective of the proposed changes within this chapter are to expand the core concepts and definitions of an IE model and subsequently, the language used to describe the structure being modelled. The aim is that by making these modifications, additional engineering knowledge and design choices regarding the structure become possible to embed within the model itself. Understanding why the modeller made certain choices when assembling the model is crucial to achieving the goal of being able to determine which structures are similar within the network.

(a) A photo of an example beam and slab bridge in Northern Ireland.



(b) An Attributed Graph representation of Fig. 3.1a when using only one element per span.



(c) An Attributed Graph representation of Fig. 3.1a using two elements per span because of sensor localisation or damage localisation requirements.

Figure 3.1: Two possible Attributed Graph representations of the same simplified beam and slab bridge. Ground is represented via a G, an element via E, boundary conditions via BC and joints via a J.

Figure 3.1 outlines one such problem when design choices of the modeller are not available within the model itself. If one imagines a simple beam and slab bridge (see Figure 3.1a) in the centre of a valley. In a simplified version, there is a single column in the centre of the bridge and then a single beam on either side, creating a path from one side of a valley to another. There are only three physical components to be included within the IE model; however, the modeller may decide to split each beam into two elements as they desire to know if the damage within the beam is towards the centre of the bridge or towards the valley side of the bridge. Damage localisation is a valid concern within the foundations of SHM and as such, needs to be honoured in any PBSHM theory.

The problem with honouring these foundational principles of SHM, is that in the scenario described above, two distinct IE models and subsequently AG's are

generated for a single unique structure (see Figure 3.1b and 3.1c). The problem is compounded when one considers such effects as how different communities see interest in the structure; one modeller may refer to a component as a 'support' and the other a 'column', one community may consider an IE model only to be of interest when at rest on the ground, another community may only consider a structure interesting when the structure is in flight. Any of the aforementioned different view points would undoubtedly change the topology of the associated IE model.

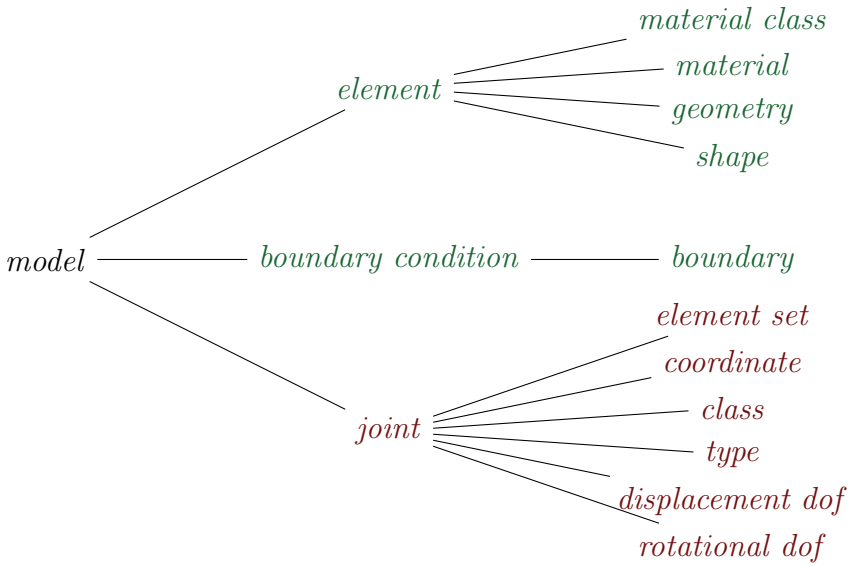


Figure 3.2: The original hierarchical data models used for describing an Irreducible Element model as described by Gosliga *et al.* [6, 38].

The aspiration of the PBSHM ecosystem (see Chapter 4) and the language of IE models, is that regardless of these design choices or view points, if an IE model is generated from the same structure, the network should be able to identify these IE models as being equivalent. Whilst this goal is theoretically achievable, within the context of the current language of IE models, there is no available method for embedding this knowledge within the model, and as such, a reconstructed language, definitions and concept of an IE models is proposed. To achieve this new concept and language of IE models, the theory put forth by Gosliga *et al.* (see Figure 3.2) is expanded upon, restructured, defined and unified into the PBSHM Schema (see Section 4.3) enabling a standardised representation of structures within PBSHM and the network of structures. The rest of the sections within this chapter, provide the definitions and language for the newly reconstructed IE model concept.

3.2 Root Objects

In an effort to simplify the language of an IE model in conjunction with enabling future expansion for yet unknown features of the language, the syntax of an IE model has been restructured. Any component included in an IE model is an *element*, and any interactions between these *element*s is a *relationship*. Subsequently, direct mapping from an IE to an AG is supported, for use within the network of structures; each *element* becomes a vertex and each *relationship* becomes an edge. Any properties associated with a *root object* — an *element* or a *relationship* — become embedded attributes on the associated vertex/edge. To denote different classifications of *object* within a *root object*, further subdivision is facilitated via an *object type*.

The subdivision of *root object*s via a type, enables a hierarchical approach to data categorisation within the model. Known scenarios of an object are associated with a type, whilst as-yet unknown scenarios are provided a vehicle for registering future scenario in the model without invalidating any existing scenario's logic. See Figure 3.3 for details on the properties available for all *root object*s. In the interest of clarity, further documentation within this chapter will use the annotation of [type] *object*, where [type] refers to the sub category/type of the named object: *object*.

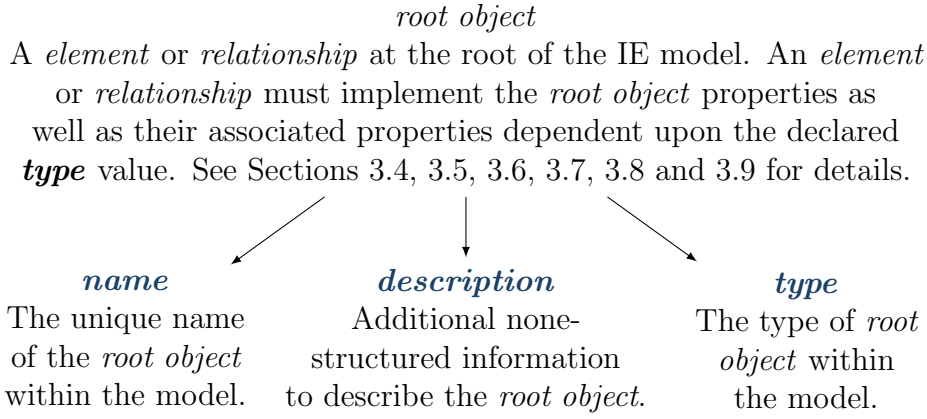
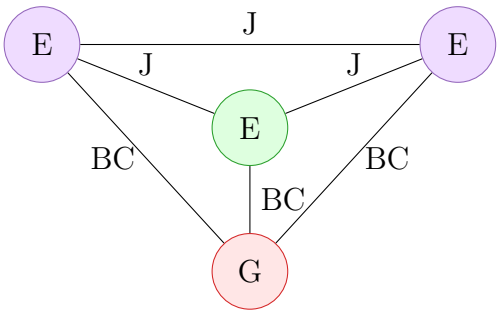


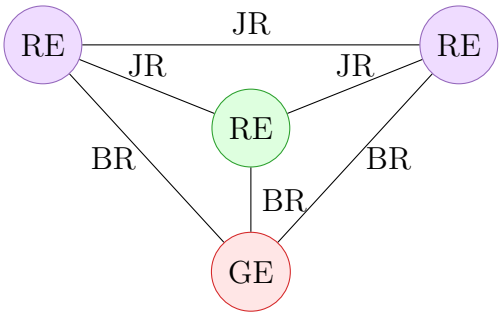
Figure 3.3: The properties for a *root object* in an Irreducible Element model.

Whilst the restructuring of *root object*s is critical for providing space within the language for the previously-stated enhancements, historical IE models must remain valid within the revised language via a minor translation. Historical components *element* and *ground* become [regular] *element* and [ground] *element*

respectively with historical interactions *joint* and *boundary condition* becoming [joint] *relationship* and [boundary] *relationship* respectively. Figure 3.4 depicts the minor translations required for historical IE models by, translating the single element per span example from the beam and slab bridge used in Figure 3.1, into the proposed expanded syntax.



(a) The AG when using the IE model language introduced by Gosliga *et al.* [6]. Ground is represented via a G, an element via E, boundary conditions via BC and joints via a J.



(b) The AG when using the IE model language proposed within this chapter. [ground] *element*s are represented via GE, [regular] *element*s via RE, [joint] *relationship*s via JR and [boundary] *relationship*s via BR.

Figure 3.4: The single element per span Attributed Graphs (AG) of the bridge displayed in Figure 3.1a using both the language introduced by Gosliga *et al.* in Figure 3.4a and the new language proposed within this chapter, Figure 3.4b.

3.3 Free and Grounded Models

The first significant step in removing ambiguity from a IE model is understanding the context in which the IE model was generated; should the IE model be considered as [grounded] to a surrounding influence or should the IE model be considered as [free] from all surrounding influences. A [free] IE model, is a model that is free from any external references within the model and must therefore, be considered as if the model was floating in a vacuum. A [grounded] IE model, is a model that references any external systems which interact with the model, whilst not containing any pertinent information to the extent which these external influences have upon the model, they merely denote that an unknown external system interacts with the modelled system at a specified intersection.

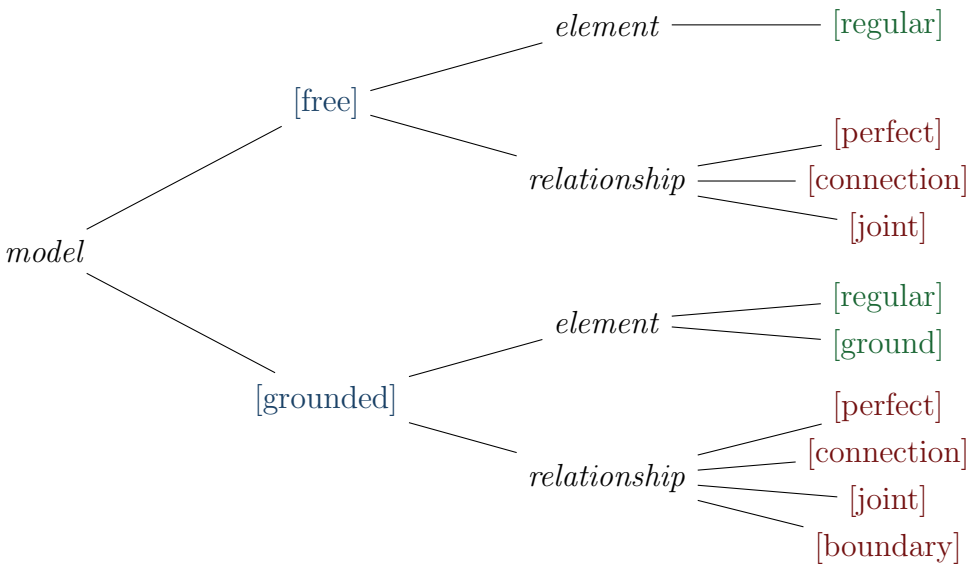


Figure 3.5: The hierarchical data models used in the standardised version of Irreducible Element models. The above diagram depicts which *element* and *relationship* types are available for a given *free* or *grounded* model.

External systems are represented within a [grounded] model via [ground] *element*s (see Section 3.4) and [boundary] *relationship*s (see Section 3.9), these types of *root object* are not permitted within a [free] model. A [grounded] model must be a valid [free] model when all [ground] *element*s and [boundary] *relationship*s are removed from the model. A [free] model must be a valid [grounded] model when all [ground] *element*s and [boundary] *relationship*s are included for a models given surrounding influences. Figure 3.5 depicts the permitted *element* and *relationship* types for the given [grounded] and [free] models.

3.4 Ground Element

A [ground] *element* represents a system which is external to the current structure being modelled. Practically, this could be as simple as a reference to the ground which a structure is placed upon, or it could reference another structure which potentially has its own IE model, both scenarios are valid within the same [grounded] model.

If one imagines two skyscrapers with a skybridge between them, there are three IE models generated and two distinct perspectives. From the skyscraper's perspective,

there will be a [ground] *element* for the foundations on which the building is built. From the skybridge perspective, there will be a [ground] *element* for each skyscraper connecting onto the bridge, but no [ground] *element*s are required for a interaction with the earth as the skybridge's interactions with the earth are via the skyscrapers.

In the currently-proposed solution within this chapter, a [ground] *element* does not have any additional *object* properties over the base *root object* (See Figure 3.3); however, in the future, additional [ground] *element* attributes could be declared to include a reference to the external system being denoted, if the system in question, has an associated IE model.

3.5 Regular Element

A [regular] *element* represents a structurally-significant component within the structure being modelled. Depending upon the purpose of the model, this could range from a large I-beam supporting the deck of a bridge, all the way down to a washer used on a bolt. It is important to note that one single component within a structure cannot be both a [regular] *element* and a [ground] *element* within the same model; however, it may be a [regular] *element* within one model and a [ground] *element* within another model.

If one imagines again the example used within the [ground] *element* (see Section 3.4) description, of a skybridge between two skyscrapers. In the two skyscraper IE models, there will be [regular] *element*s representing the structurally-significant components; however, some of these same components may be declared as [ground] *element*s within the IE model of the skybridge, as they provide the support on which the skybridge rests.

As a [regular] *element* represents a physical component within the structure, there are additional details over the base *root object* that need to be captured in order to embed the physical and structural significance of this *element* within the model. To achieve the aforementioned embedding of data within the model, there are four significant types of details to be captured via *child object*s: coordinates, context, geometry, and material (see Sections 3.5.1, 3.5.2, 3.5.3 and 3.5.4 respectively). Figure 3.6 depicts the hierarchical layout of a [regular] *element* and the corresponding *child objects*.

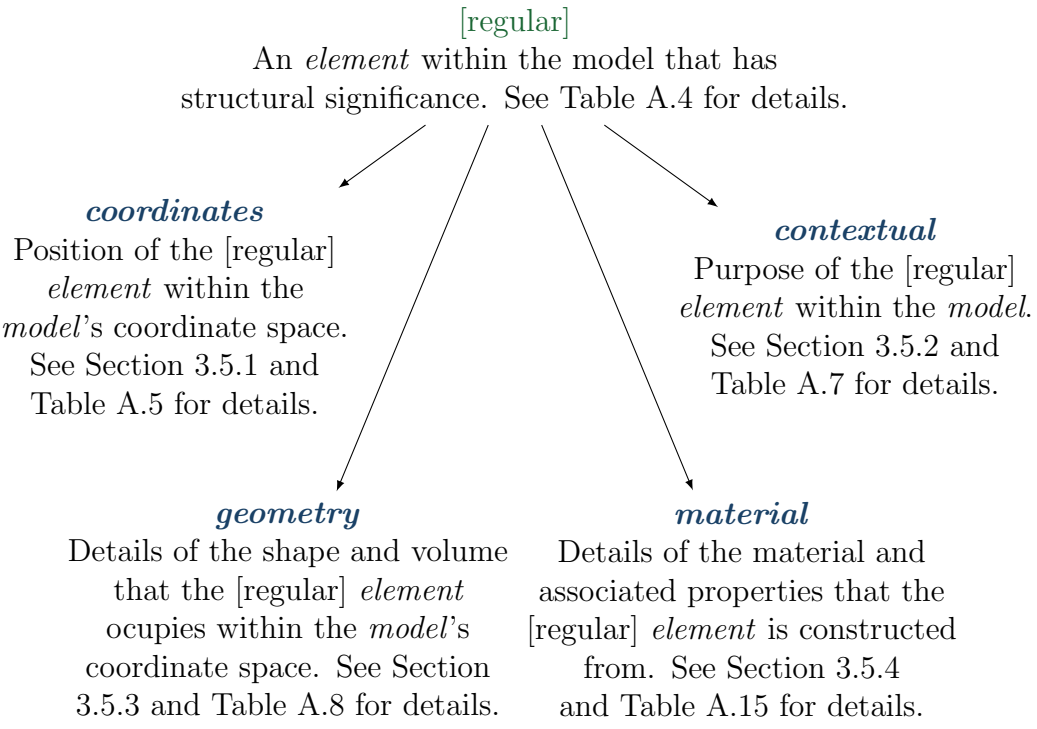


Figure 3.6: The hierarchical layout of child objects within a [regular] *element*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table A.4.

Within the *network*, there will inevitably be structures of different forms, compositions, and purpose. Adding additional areas of knowledge into a [regular] *element* enables a granular approach for determining the homogeneity of structures. Differing user scenarios may necessitate contrasting requirements for determining homogeneous structures within the network. User A may determine that only a shared abstract form is required, whilst user B may require that not only is there a shared abstract form, but the structures must share a common composition.

Depending upon the *child object* of a [regular] *element* in question, will determine whether the aforementioned *child object* is required or not. The *child object*s are designed to support the rich set of data required to embed engineering knowledge and design decisions within themselves; however, it is understood that for many structures, this knowledge and decisions may not be available, as such, if a *child object* is required for a valid [regular] *element* the minimum data required for a *child object* to be valid is the ***type*** property.

3.5.1 Coordinates

The purpose of the *coordinates* object within a [regular] *element* is to facilitate capturing the required data to represent the position of an element within the 3D coordinate space of an IE model. Whilst capturing the position of the [regular] *element* is optimal, this may not always be possible given the scale and age of potential structures within the *network*; as such, providing a *coordinates* object for a [regular] *element* is not required. Figure 3.7 displays the hierarchical properties within a *coordinates* object and Table A.5 gives a full list of all the properties, including which properties are required.

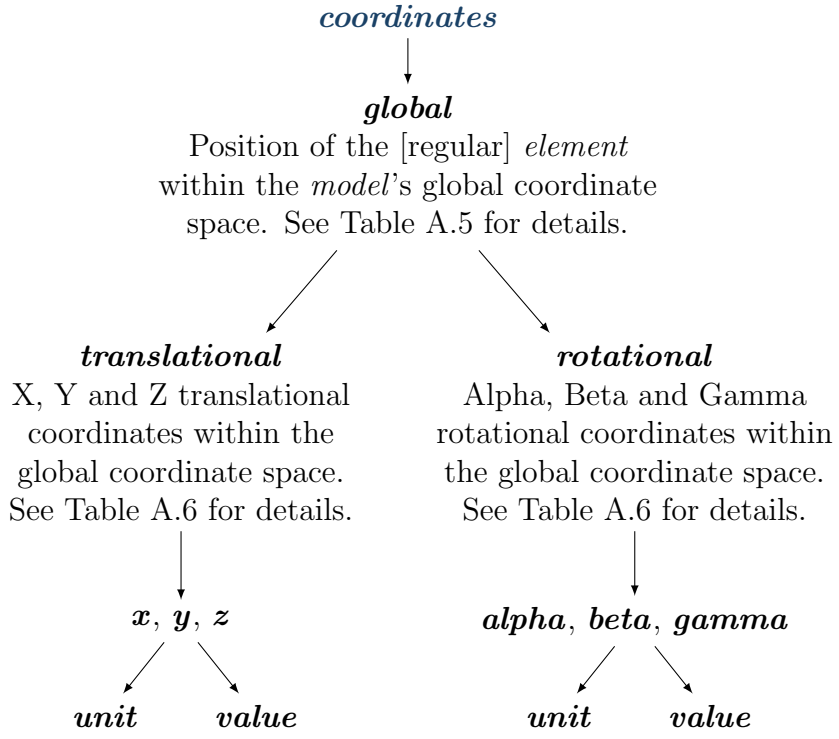


Figure 3.7: The hierarchical layout of child objects within the *coordinates* section of a [regular] *element*.

Throughout an IE model, there is a generalised theme of capturing the position of the item in question within the 3D coordinate space. Whilst the current enclosed solution only implements a global coordinate space, it is acknowledged that there should be provision for supporting multiple coordinate spaces within the model. As such, the only direct property within the *coordinate* object declares that all further information is within the global coordinate space. By structuring the *coordinates*

object in this manor, the principles of organic data design are obeyed and room is left for additional coordinate spaces to be declared without invalidating existing coordinate data. Whilst it is possible to provide both the translational and rotational information within the global coordinate space, only the translational information is required for the *coordinates* object to be valid.

In the interest of clarity, the global coordinate space used within IE models is stated to have an origin O in the bottom left corner of an IE model, where $x = 0$, $y = 0$ and $z = 0$. The x axis operates in the horizontal direction from left ($x = 0$) to right ($x = n$), the z axis operates in the vertical direction from bottom ($z = 0$) to top ($z = n$) and the y axis operates in depth direction from near ($y = 0$) to far ($y = n$) (see Figure 3.12 for a labelled diagram).

3.5.2 Context

An important distinction between the initial IE model language and the proposed IE model language included within this chapter, is the separation of the function a component has within a structure verses the shape and volume occupied by the component. The former is modelled by the *contextual* object and the latter is modelled by the *geometry* object (see Section 3.5.3). As the *contextual* object embeds the purpose of the component into an IE model, the property is required for a valid [regular] *element*. Figure 3.8 displays the hierarchical properties within a *contextual* object and Table A.7 gives a full list of all the properties including which properties are required.

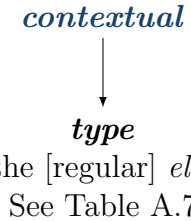


Figure 3.8: The hierarchical layout of child objects within the *contextual* section of a [regular] *element*.

As the knowledge included within a *contextual* object describes the function of the underlying component, the terms used to describe this function should be appropriate to the given modelling audience. Currently, this knowledge embedding

is supported via the declaration of a flat ***type*** property where a value can only be selected from a list of supported types. It is expected that via the production of diverse structure-type IE models; additional types will be realised and included within the allowed types.

3.5.3 Geometry

The *geometry* object, in partnership with the *coordinates* object (see Section 3.5.1) enables capturing the form of a structure. Understanding the form of a structure is an integral part of determining the homogeneity of structures, as such, the *geometry* property is required for a valid [regular] *element*. Figure 3.9 displays the hierarchical properties within a *geometry* object and Table A.8 gives a full list of all the properties, including which properties are required.

Whilst the *coordinates* object is not required for a valid [regular] *element*, if any dimensions are provided within the *geometry* object, it is highly recommended that a *coordinates* object is generated — albeit with nominal values — otherwise any inferred shape has no point of reference.

The primary embedding of knowledge within the *geometry* object is via the type. Unlike the flat type used within the *contextual* object, a layered hierarchical forest approach is used within the *geometry* object to embed the initial knowledge as a type. Each tree within the forest is a rooted tree with a directed graph in the direction from the root node to the leafs. The selected type for the *geometry* object must be a complete path from a root node to its corresponding leaf node.

The reasoning behind a hierarchical approach for the type is clear when considering how the geometrical type may be used within determination of the homogeneity of structures. If one imagines two structures being compared for similarity; one [regular] *element* may be declared as a rectangular beam (***beam*** → ***rectangular***) in one structure and as a I-beam (***beam*** → ***i-beam***) in another model — because of the available knowledge. It may be considered appropriate, especially if one was primarily concerned about the *material* composition, to say that the *geometry* properties 'matched' if the root node within the type tree was the same, e.g. if type layer 0 was ***beam***. Having this hierarchical approach to the type within the *geometry* object, enables approximate similarities within future matching algorithms.

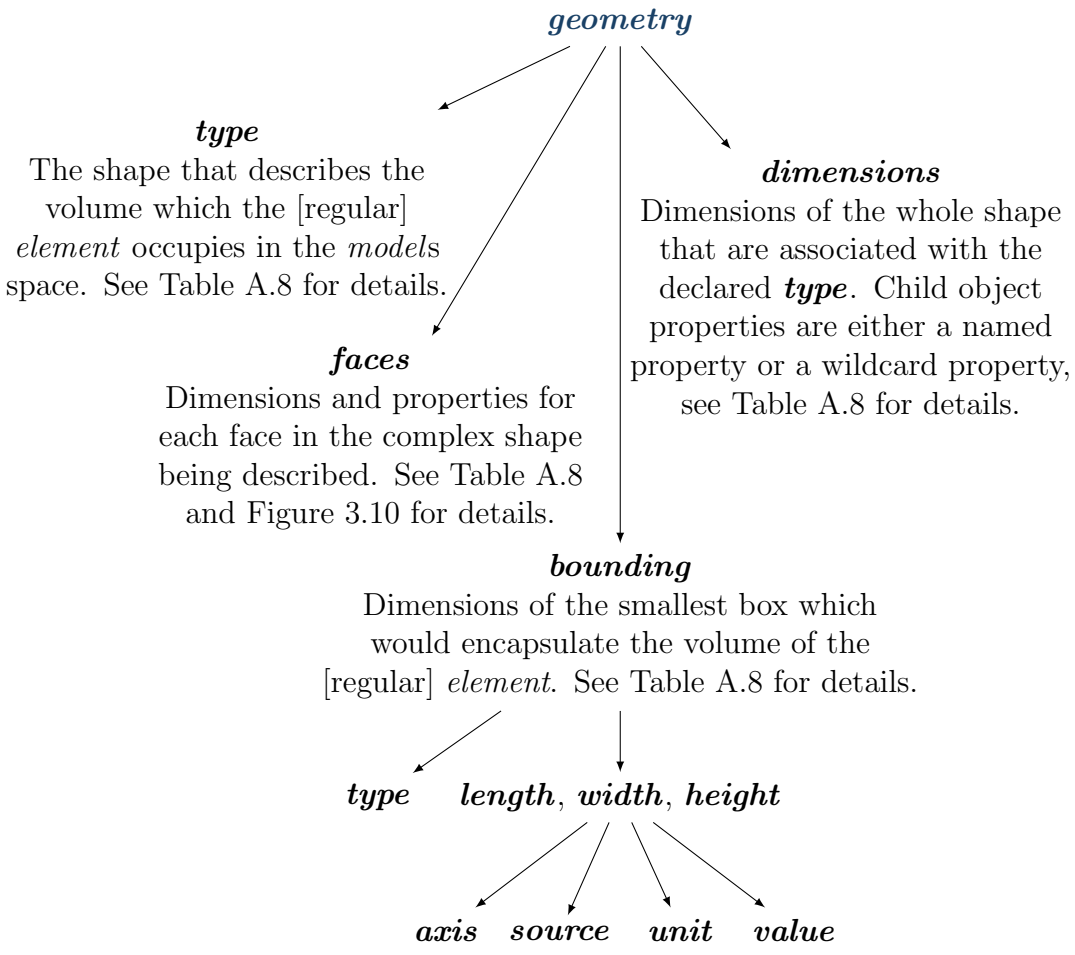


Figure 3.9: The hierarchical layout of child objects within the *geometry* section of a [regular] *element*.

For simplistic shapes which can be represented by a commonly-known name (see Table A.9), associated dimensions are required dependent upon the type selected — if any dimensions are given at all. If one imagines selecting a rectangular beam (***beam*** → ***rectangular***) as the type, giving no dimensions is still valid; however, if any dimensions are provided, a ***length***, ***width*** and ***height*** property must be declared as property dimensions as a minimum. Whilst this simplistic approach enables capturing the form for standard geometrical shapes, it is understood that this approach will fall short for describing complex geometrical shapes (see Table A.14), where referencing a commonly-known name is either impractical or would bring confusion to the understanding of the shape.

For these aforementioned complex geometrical shapes, an abstract representation of the shape is required with further knowledge regarding the form of the shape to

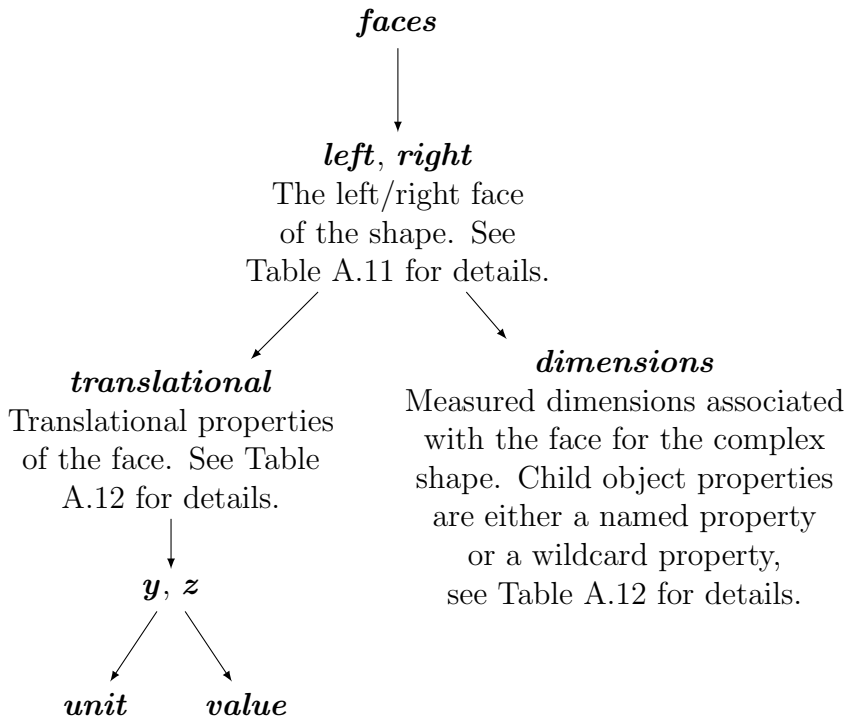
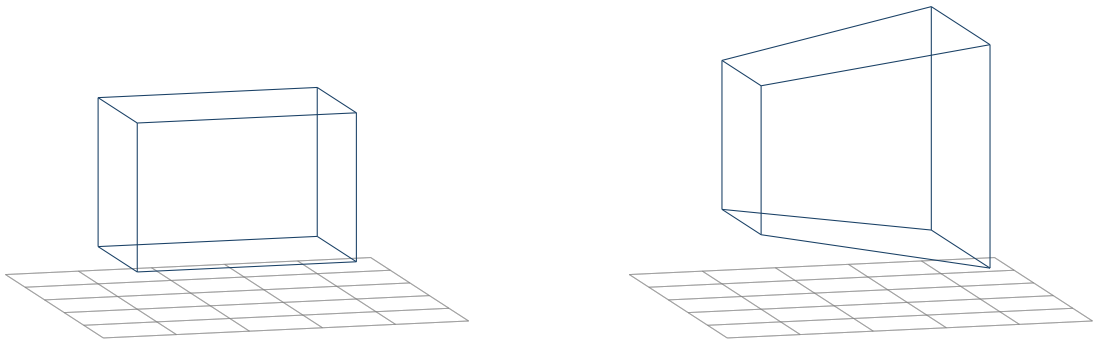


Figure 3.10: The hierarchical layout of child objects within the *faces* section of *geometry* within a [regular] *element*.

be embedded via additional *geometry* properties. These abstract representations of geometrical shapes are envisioned to be activated via supplementary type branches under the ***solid*** and ***shell*** root geometrical types. The first such abstract representation supported is via the ***translateAndScale*** branch type under both ***solid*** and ***shell*** root types.

The premise of the *translate and scale* type, is that one imagines the same two-dimensional shape – say a rectangle – on both the left and right side of a volume. The space between the two-dimensional shapes can be as large as required (the *translational* part), and the two-dimensional shape itself may also be scaled up or down in comparison to its counterpart (the *scale* part). The volume is then completed by joining corresponding vertices in the defining faces. Figure 3.11 depicts this example via two cuboid shapes, the first (Figure 3.11a) has a consistent cross-section throughout the length of cuboid, and as such can be represented via a simplistic commonly-known named geometry (***solid*** → ***translate*** → ***cuboid***), the second (Figure 3.11b) has an increasing cross-section through the left section of the shape to the right section, and would be represented via the translate and scale

complex geometrical type (***solid*** → ***translateAndScale*** → ***cuboid***).



(a) A square box represented as a standard commonly-known named representation; ***solid*** → ***translate*** → ***cuboid***

(b) A tapered square box represented as the complex geometry representation; ***solid*** → ***translateAndScale*** → ***cuboid***.

Figure 3.11: The difference between a standard commonly-known named representation of a components geometry and a complex representation of a component geometry.

When using an abstract geometrical type, the aforementioned imaginary volume, is modelled using the ***bounding*** and ***faces*** properties of the *geometry* object. The ***bounding*** and ***faces*** properties are only available for a population, once an abstract geometrical type is selected, and are required to be valid. The principle of the ***bounding*** property is that one is describing the smallest shape in which the component — or section of component — could be placed and the bounding completely encapsulate the component. The principle of the ***faces*** property is to describe the geometrical properties of the two-dimensional shape upon the face (*the left and right of the imaginary space in the Figure 3.11 example*).

Currently, only cuboid bounding boxes are supported, which necessitates the requirement of a ***faces*** property to embed the knowledge regarding the form of a component; however, in the future, yet-unknown bounding boxes may be included which support the capturing of a component's form in a different manner than the declaration of faces. Figure 3.12 depicts the interactions between the component being modelled as a [regular] *element* and the ***bounding*** and ***faces*** properties of the *geometry* object.

For a cuboid bounding, it is said that the component in question, must transverse through the bounding in a horizontal direction (x axis), as such, whilst there are

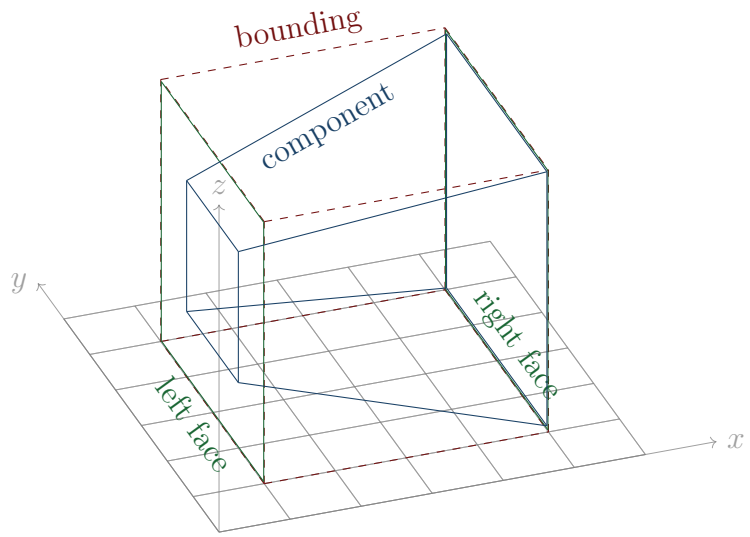


Figure 3.12: The interaction between a component's complex geometrical shape and the associated bounding box and faces.

6 faces to a cube, only the left ($x = 0$) and right ($x = n$) faces are of interest. If a component's primary direction of transversing through the cuboid bounding is in either the y or z axis, the bounding must be rotated via the *coordinates* object as such, so that the primary direction through the bounding, is through the x axis.

Each face in question, captures dimensional properties for the component at the cuboid bounding's given position within the global coordinate space, dependent upon the selected abstract type (see Table A.14). If a cuboid bounding encapsulates the component in its entirety, then the dimensions captured on each face must be considered as if there is no space on the x axis between the face and component. If the cuboid bounding only encapsulates a section of the whole component, then the dimensions captured for each face must be considered as if the face slices the component at the given position. Additionally, if the slice made in a component does not fill the face causing the slice, then translational movements must be captured to embed the position of the component within the face.

The axis values for the ***length***, ***width***, and ***height*** properties of the cuboid bounding, must match the direction of travel of the corresponding axis within the global coordinate space. For example, the x axis is stated to travel along the horizontal axis within the global coordinate space, as such, if x is given as the value for the axis for the ***length*** property, this is stating that the length dimension for the cuboid bounding is along the horizontal axis of the global coordinate space.

The values for the translational movements within a face are local coordinates to the face, with the axis and direction of travel matching the global coordinate space; as such, only y and z coordinates are able to be captured. Translation movement values should be based around moving the centroid of shape within the face. For instance, in Figure 3.12 there is a component on a 6×6 grid. The type of the *geometry* would be said to be ***solid*** $\rightarrow$ ***translateAndScale*** $\rightarrow$ ***cuboid***. The *coordinates* of the component would be stated as $x = 1$, $y = 1$ and $z = 0$. The cuboid bounding is an even cuboid with 4 units in each dimension. On the left face, there is a 2×2 square caused by the slice, perfectly in the centre of the face. On the right face, there is a 4×4 square, completely filling the face as this is the end of the component.

The left face would be stated as having width (y axis) and height dimensions (z axis) with a value of 2, with the translational values being $y = 2$ and $z = 2$; this would position the shape in the left face at; $x = 1$, $y = 3$ and $z = 2$ within the global coordinate space. Conversely, the right face is stated so that the width and height dimensions completely fill the face, as such, each dimension has a value of 4 and the translational values are said to be $y = 2$ and $z = 2$. This would position the shape in the right face at; $x = 5$, $y = 3$ and $z = 2$ within the global coordinate space.

3.5.4 Material

Whilst comprehending the form of a component is a key factor in determining the behaviour and similarity of structures, gaining the knowledge of what components are comprised of, is another fundamental influence in understanding the dynamics of structures. The *material* object embeds this very compositional knowledge of a component, into a [regular] *element*. As the composition of a structure may be a determining factor of whether two structures are homogeneous, the *material* object is required for a valid [regular] *element*. Figure 3.13 displays the hierarchical properties within a *material* object and Table A.15 gives a full list of all the properties, including which properties are required.

Similar to the other [regular] *element child object*s, the primary method of embedding data is via the ***type*** property (see Figure A.1). Whilst the type used within the *material* object is a hierarchical forest — like the *geometry* object — the selected type does not have to complete a path from a root node to a leaf

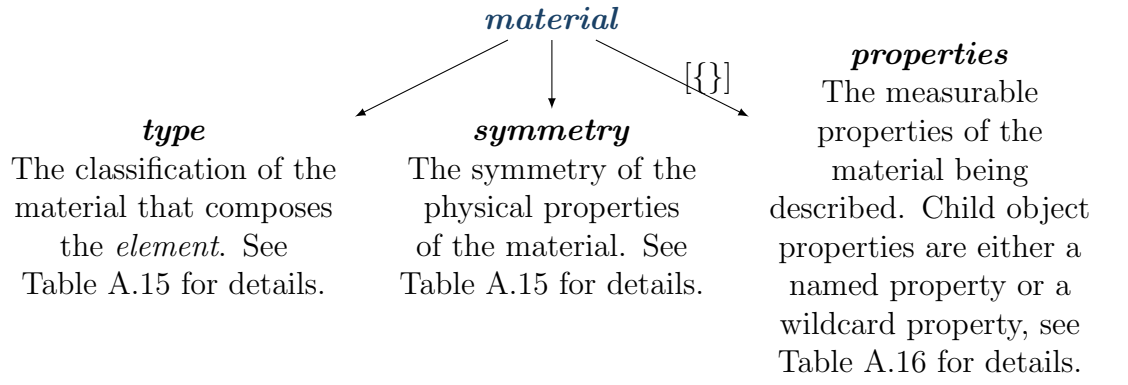


Figure 3.13: The hierarchical layout of child objects within the *material* section of a [regular] *element*. A note on the [{}] symbol; this is used on the arrows to denote when the property it is pointing towards accepts an array of objects, instead of a singular value or object.

node, the path may terminate at any corresponding valid child node given the selected root node.

The ***type*** provides both a simple categorisation of material but also an implied understanding of materialistic information; however, this implied understanding does not facilitate the granular level of detail required within certain similarity determinations. Additional material properties of a [regular] *element* can be embedded within a *material* object via the ***properties*** property.

Because of the available information at the time of generating a model, only a singular value may be known for a given material property, as such, this is the minimum required information — in conjunction with the unit, where appropriate. It is also understood, that for some material properties, there may be the rich knowledge of multiple values at given test conditions and environmental variables; *conditional value*s. For conditional values, instead of providing a singular value, a list of values is provided with associated test conditions — the ***parameters*** property — and environmental conditions — the ***environmental*** property.

To facilitate the flexibility required for the varying degree of material properties specified within a structure, three categories of named material properties are supported; material properties free from any associated units (see Table A.17), which are deemed valid for either a singular or conditional value, material properties dependent upon an associated unit to correctly quantify the value (see Table A.18), which are also deemed to be valid for either a singular or conditional value and

lastly, material properties which can only be considered valid with predetermined test conditions (see Table A.21), and as such only accept a conditional value.

For the avoidance of confusion, a *named* property is a property which is known by the PBSHM Schema and as such, is defined within the schema with a set of accepted associated properties. A *wildcard* property is a property which is yet unknown by the PBSHM Schema and as such, does not constrain what the associated accepted properties are. For instance, a material property with the type set to 'yieldStrength' has an accepted list of valid ***unit*** values as defined in Table A.18. Currently, only isotropic material properties are supported within the proposed *material* object; however, it is expected that future versions of an IE model will support anisotropic material properties.

3.6 Perfect Relationship

A [perfect] *relationship* represents the interaction between two — and only two — [regular] *element*s where the two *element*s in question could be considered to be the same singular *element*; segmentation only occurring to embed additional geometrical knowledge or enhanced localisation, within the model. The two *element*s within the *relationship* must have matching ***type*** values for the *contextual*, *geometry* and *material* objects of a [regular] *element*. Additionally, any defined properties within the *element*s must not cause any logical conflicts between themselves.

If one imagines the fuselage of an aeroplane as a component within an IE model. If one were to only use a single [regular] *element* to describe the component, vast amounts of rich geometrical knowledge would not be embedded within the model. In contrast, if multiple [regular] *element*s were instead used, the rich geometrical knowledge is successfully embedded within the model; however, the cost of implementation would be an introduced ambiguity regarding the number of *element*s into which a component should be segmented. The proposed solution for this problem, is the [perfect] *relationship*; by embedding segmentation knowledge into the model, a reduction of *element*s can occur before any similarity determinations.

In the pursuit of embedding the form of a structure within an IE model, the *relationship*s proposed within this chapter — akin to [regular] *element*s — support

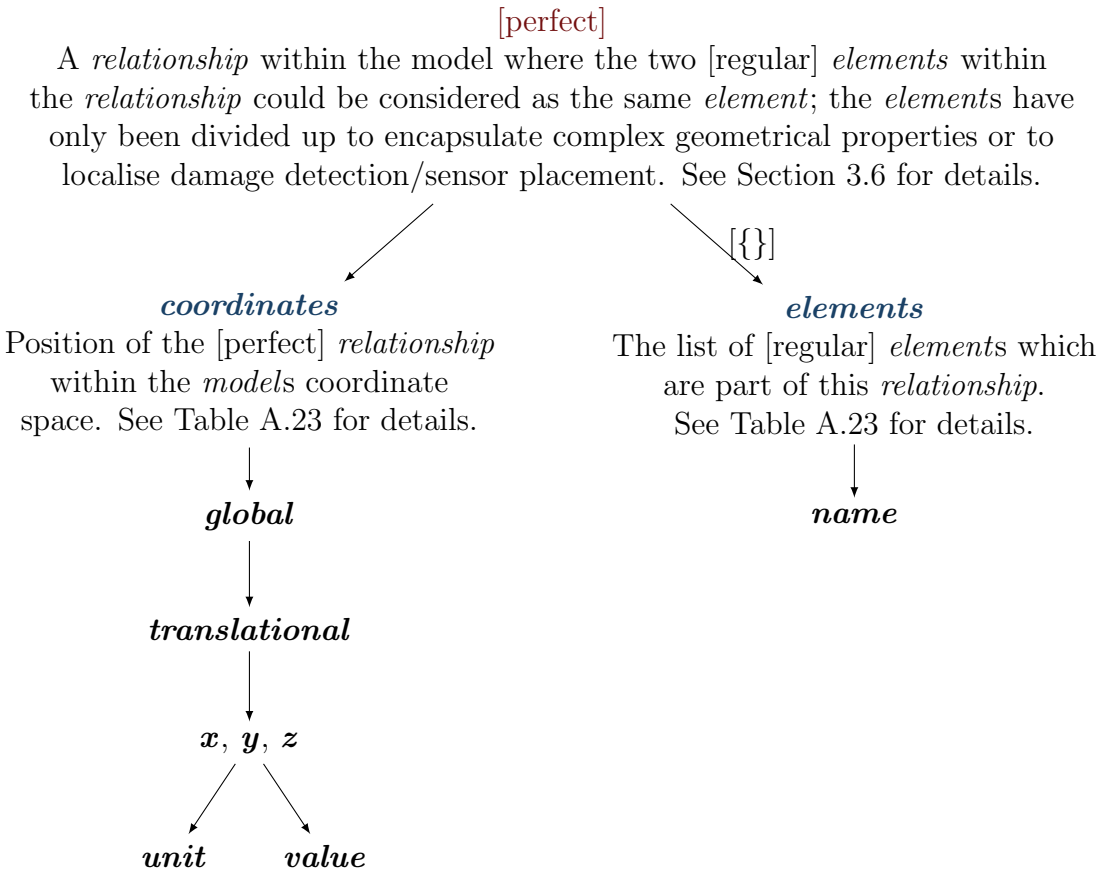


Figure 3.14: The hierarchical layout of child objects within a [perfect] *relationship*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table A.23.

the inclusion of positional data via a coordinates object; depending on the number of *element*s included within a *relationship*, it determines if singular or multiple coordinate objects are facilitated. In the interest of clarity, the coordinates provided within the *relationship* must reference the centroid of the two faces which contact, as such the coordinates provided therefore must be based within the same global coordinate space used for [regular] *element* locations.

Within the context of a [perfect] *relationship*, only two [regular] *element*s are accepted, subsequently, only a singular coordinate object is applicable for this scenario. Figure 3.14 displays the hierarchical properties within a [perfect] *relationship* and Table A.23 gives a full list of all the properties including which properties are required.

3.7 Connection Relationship

A [connection] *relationship* represents the interaction between two or more [regular] *element*s where the *element*s in question are held together by an unknown component excluded from the model, nominally because of the component having no immediate structural significance. A [connection] *relationship* may only be used within a model, if — and only if — the *relationship* can be substituted for numerous [joint] *relationship*s (see Section 3.8), without loss of knowledge within the model. The previously-omitted component is modelled by; inclusion of a new [regular]

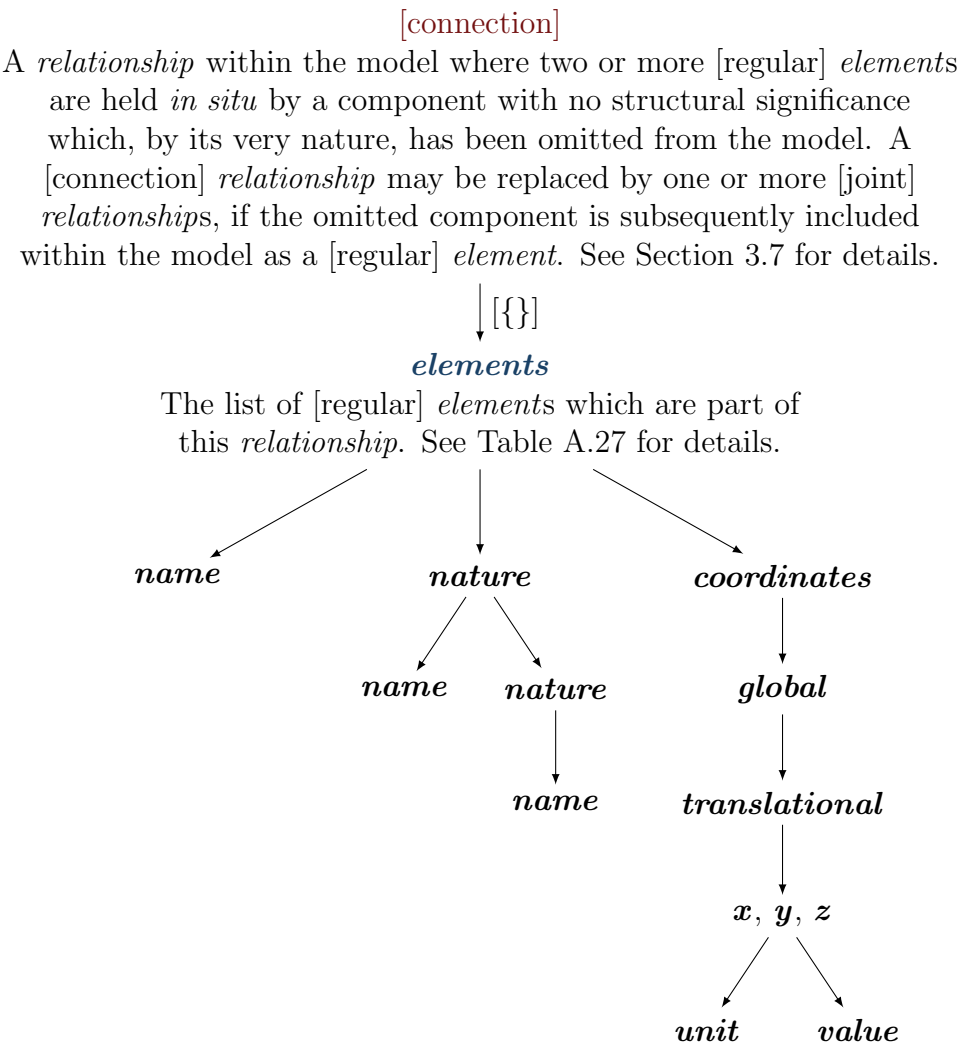


Figure 3.15: The hierarchical layout of child objects within a [connection] *relationship*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table A.27.

element, subsequently a [joint] *relationship* is created for every *element* previously included within the [connection] *relationship*, and the newly instantiated [regular] *element*.

The *relationship* must be considered static without any notable movement present within the *relationship*. If any notable movement is present within the *relationship*, a [connection] *relationship* is no longer valid and the interaction thus must be modelled as a [joint] *relationship*. The *element*s included in the *relationship* may have different properties within the [regular] *element*, without invalidating the *relationship*.

If one imagines a section of a bridge where three beams connect together; there is a singular beam running horizontally, with two other beams at 45 degree angles from the horizontal beam (see Figure 3.16). All the beams connect together in the centre of the horizontal beam. There is also a face plate covering the connection of all of these beams. One could make an argument for stating that the face plate is not structurally significant within the structure; as such, the face plate could be ignored from the model and the interaction between the three beams modelled as a [connection] *relationship*, which positions them rigidly in the correct spatial relation to each other. Conversely, an argument could also be stated for the inclusion of the face plate, thus requiring the interaction to be modelled using multiple [joint] *relationship*s.

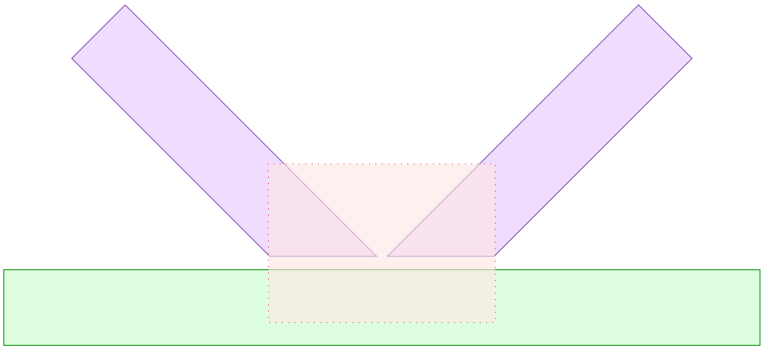


Figure 3.16: An example of where a [connection] *relationship* could be used within an IE model. In this example, there are three beams all connecting together — the rectangles in purple — in the centre, with a face plate — the dotted red rectangle — covering the connection. This interaction can be modelled as a [connection] *relationship* between all three of the beams, by omitting the face plate from the IE model. This modelling methodology, is only possible using the premise that the face place has no structural significance within the model; however, if the face plate is stated to have structural significance, the interaction must therefore be modelled as individual [joint] *relationship*s, between each beam and the face plate.

Coordinates are supported within a [connection] *relationship*; however, because of the nature of a [connection] *relationship*, allowing the ability for each [regular] *element* within the *relationship* becoming its own [joint] *relationship*, coordinate data therefore must be stored against each *element* within the *relationship* instead of against the overarching *relationship*. Figure 3.15 displays the hierarchical properties within a [connection] *relationship* and Table A.27 gives a full list of all the properties including which properties are required.

3.8 Joint Relationship

A [joint] *relationship* represents the interaction between two — and only two — [regular] *element*s where the physics of the interaction between the two *element*s is desired to be modelled. If the associated scenarios for a [perfect] or [connection] *relationship* are not applicable for the given [regular] *element*s then a [joint] *relationship* should be the selected *relationship*.

Both a [perfect] and [connection] *relationship* may be modelled as a [joint] *relationship* — with a given [static] *nature* — and remain a valid [joint] *relationship*; however, it is strongly encouraged that an interaction between two [regular] *element*s which adhere to either the scenarios outlined in a [perfect] or [connection] *relationship* are declared as a [perfect] or [connection] *relationship* for the additional implicit context the definitions give to the *relationship*.

To correctly encapsulate the physics of the interaction within a [joint] *relationship*, additional subdivision is facilitated via the *nature* of the [joint]. A [joint] *relationship* is stated to have a [static] *nature*, if both *element*s within the *relationship* have no movement between the two *element*s. A [joint] *relationship* is stated to have a [dynamic] *nature* when, within the *relationship*, there is knowledge regarding kinematic or dynamic degrees of freedom (DOF) between the two *element*s which thus, could be embedded within the *relationship*.

Whilst implicit knowledge regarding the available movements within the *relationship* is gained via knowing if the *nature* is [static] or [dynamic], explicit knowledge can be embedded within the *relationship* via declaring ***translational*** and ***rotational*** properties within the ***degreesOfFreedom***. The ***degreesOfFreedom*** property is only available within a [joint] *relationship* when the *nature* of the [joint] is [dynamic].

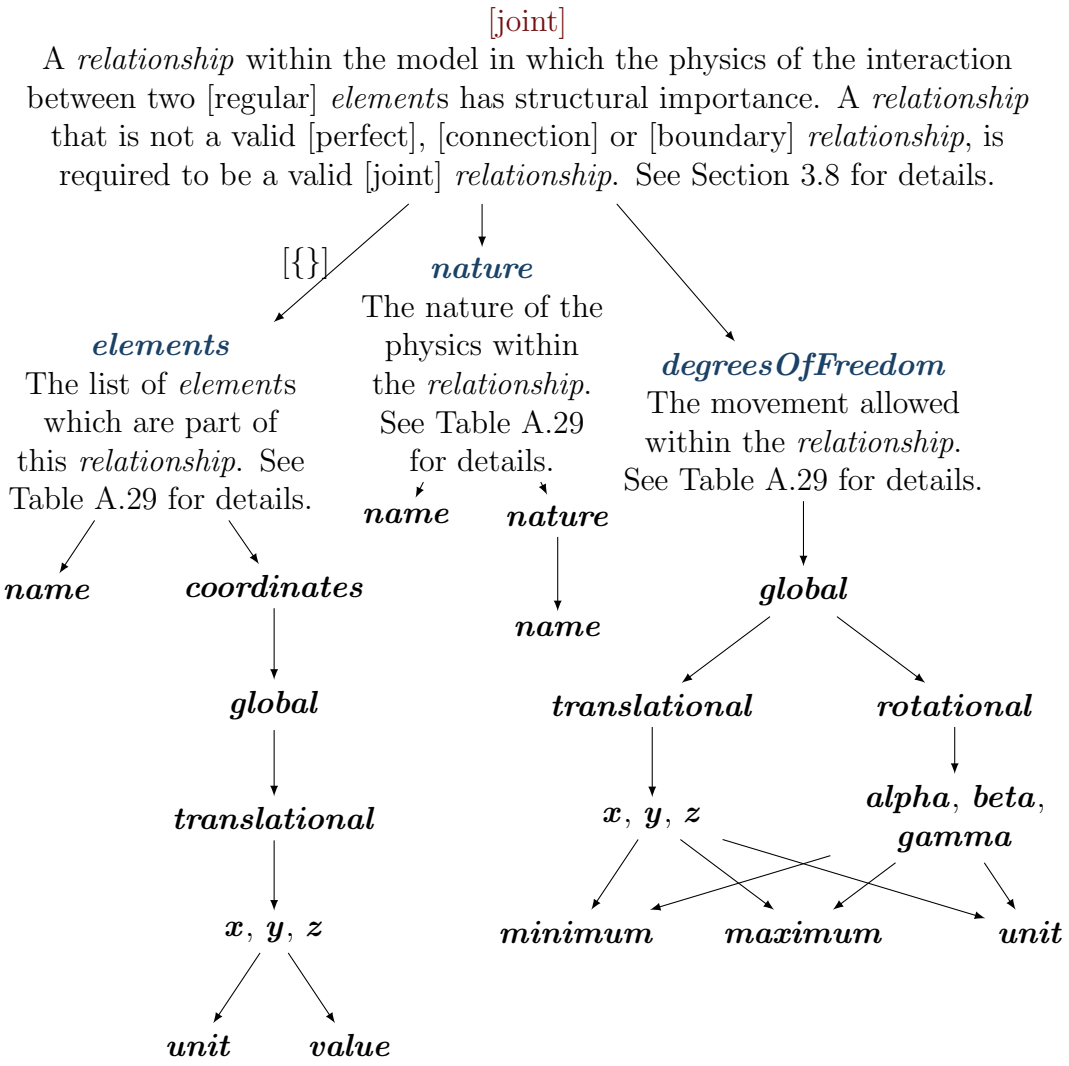


Figure 3.17: The hierarchical layout of child objects within a [joint] *relationship*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table A.29.

If one again imagines the bridge example given as part of the [connection] *relationship* (see Section 3.7). The face plate *element* has been introduced within the model and as such, the [connection] *relationship* is being remodelled as [joint] *relationship*s. One may determine that both theoretically and practically, there should be no movement between the face plate and the beam and as such, this could be modelled as a [static] ***nature*** [joint] *relationship* — ignoring the fact that a [connection] *relationship* being remodelled as [joint] *relationship*s must in fact be of a [static] ***nature*** to be a valid [connection] *relationship* — using 'static' → 'welded', 'static' → 'bolted', 'static' → 'adhesive' or 'static' → 'other'.

Coordinates are further supported within a [joint] *relationship*; however, given that there are only two [regular] *element*s included in the *relationship*, a singular coordinate object only is facilitated. Figure 3.17 displays the hierarchical properties within a [joint] *relationship* and Table A.29 gives a full list of all the properties including which properties are required.

3.9 Boundary Relationship

A [boundary] *relationship* represents the interaction between two — and only two — *element*s, where the first *element* within the *relationship* must be a [regular] *element* and the second *element* within the *relationship* must be a [ground] *element*. A [boundary] *relationship* denotes the boundary between the structure being modelled and any system which can be considered as external to the modelled structure (see Section 3.4).

By the same premise as a [ground] *element*, a [boundary] *relationship* only contains the pertinent information to denote an external system's presence within the model, as such, the only information captured within a [boundary] *relationship* is the position of the relationship within the model's global coordinate system and the name of the two *element*s within the *relationship*.

If one imagines a stationary aeroplane resting on the tarmac of an airport runway; one could state that a [grounded] IE model of the aeroplane would best represent the current state of the structure. [Regular] *element*s would be included for the standard sections in the structure; fuselage, wings, landing gear, etc. At least one [ground] *element* will need to be included within the model, to represent the airport runway which is supporting the plane. The interaction between the wheels of the aeroplane — [regular] *element*s — and the airport runway — [ground] *element*s — would be modelled by multiple [boundary] *relationship*s — one for every aeroplane wheel being supported by the airport runway — within the aeroplanes' [grounded] IE model. Figure 3.18 displays the hierarchical properties within a [boundary] *relationship* and Table A.23 gives a full list of all the properties including which properties are required.

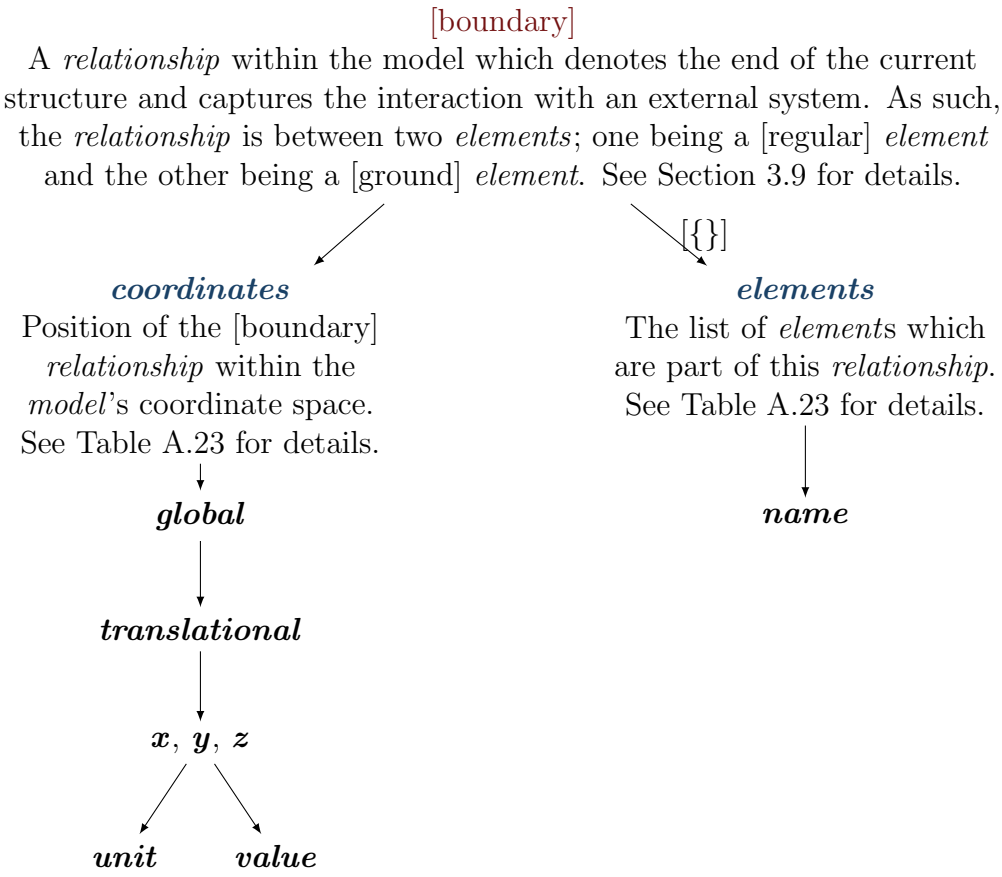


Figure 3.18: The hierarchical layout of child objects within a [boundary] *relationship*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table A.23.

3.10 Conclusion

This chapter provides a reconstruction of core concepts, definitions, and language of an IE model, with the aim to provide a standardised representation of structures and aid towards generation of accurate similarity metrics within PBSHM. The new IE model language unifies any physical component within the model as an *element*, and any interactions between these components as a *relationship*. Each *element* and *relationship* supports multiple use-case scenarios via the introduction of a *type*. An *element* can be sectioned into two supported types: a [regular] *element* for when a component is part of the structure being modelled, and a [ground] *element* when a component should be considered as external to the current structure being modelled.

A *relationship* can be sectioned into four supported types: a [perfect] *relationship*,

when the two [regular] *element*s are considered as part of a larger single [regular] *element* and have only been divided to encapsulate a components' complex geometrical form; a [connection] *relationship* where two or more [regular] *element*s are held together by an unknown structurally-insignificant component which has been excluded from the model; a [joint] *relationship* where the physics between two [regular] *element*s is included within the model, and a [boundary] *relationship* to denote where a [regular] *element* and a [ground] *element* come in contact with each other and indicate the boundary of the structure.

To enable the understanding of the structure within the outside world, the model itself may also be divided into two type: a [free] *model*, where the structure is considered in a void — no [ground] *element*s or [boundary] *relationship*s may be present within the model — or as a [grounded] *model* where the interactions between the structure and any external systems are included in the model. Each type of object within the model, has its own associated domain of knowledge with a list of accepted attributes and values to facilitate a fair embedding of knowledge on the structure.

Chapter 4

Network, Framework, & Database

This chapter follows on from the previous chapter by providing the underlying technical implementation for a shared-data domain in PBSHM. Being able to describe the structures within PBSHM in a standardised and equal format is only part of the solution, there must also be a domain in which PBSHM data can reside for the purpose of similarity comparisons. In the wider context of PBSHM, there will be vast amounts of data which are requiring processing, to facilitate the highlighted goals of PBSHM.

The data-processing requirements of an IE model are only a small portion of the total data requirements of PBSHM; as such, this chapter introduces a shared-data domain to encapsulate all underlying PBSHM data in a singular format and residency via the following themes; *network* — the shared domain in which the relationships between structure reside, *framework* — the shared domain in which PBSHM specific algorithms and computations reside, and *database* — the shared domain in which all PBSHM data reside for use within the framework and network as appropriate.

This chapter follows the following narrative: Section 4.1 outlines the current state of a shared domain with PBSHM. Section 4.2 sets out the definitions of the *network*, *framework*, and *database*, Section 4.3 describes the technical implementations constructed for the *database* and *framework*, and Section 4.4 shows the *framework* computing similarity scores for the included datasets.

4.1 Background

In a traditional (SHM) scenario, one would typically be considering the health state of a single structure by utilising data collected from the structure in question. From a data point of view, this would typically mean that any researcher or engineer trying to establish the health of a structure, would only have a single dataset to process. Whilst this theoretical dataset, may include multiple formats, they are typically limited in their variations within the dataset [15].

In contrast, PBSHM must, by its own definition, deal with data from multiple structures and consequently, a never-ending variation in formats, languages, and methodologies. These potential variations within PBSHM, cause problems when attempting to build PBSHM-specific algorithms. If a potentially-infinite number of formats were to be supported within PBSHM, this would necessitate a never-ending cycle of updating PBSHM algorithms to understand these new data formats. This process would be both impractical and commercially unviable for any real-world adoption of PBSHM technology.

Therefore, PBSHM requires a new standardised methodology towards data storage that addresses the unique challenge of accommodating multiple structure types and varying data requirements. This shared standard must facilitate a uniform way of describing and storing data on structures, support the flexibility of different data types and enable easy identification of structures within a population or type. It is important to acknowledge that structure owners may have existing systems in place to monitor and capture SHM data; in order for industrial adoption and adaptation, the standard must work in conjunction with existing systems and not enforce replacement of existing systems.

There are two main methodologies for storing of data; flat files — such as Comma-Separated Values (CSV) and plain text — and databases. Whilst a flat file structure facilitates the viewing of data as a trivial task, querying and building relationships between these data can become difficult when comparing data across multiple files. In comparison, when using a database, querying and comparing data becomes easy, whilst importing data can become a more difficult task, especially when importing large amounts of historic data. As a key principal of PBSHM is to facilitate increased knowledge retrieval, having a data store that allows data to be retrieved easily and efficiently is key.

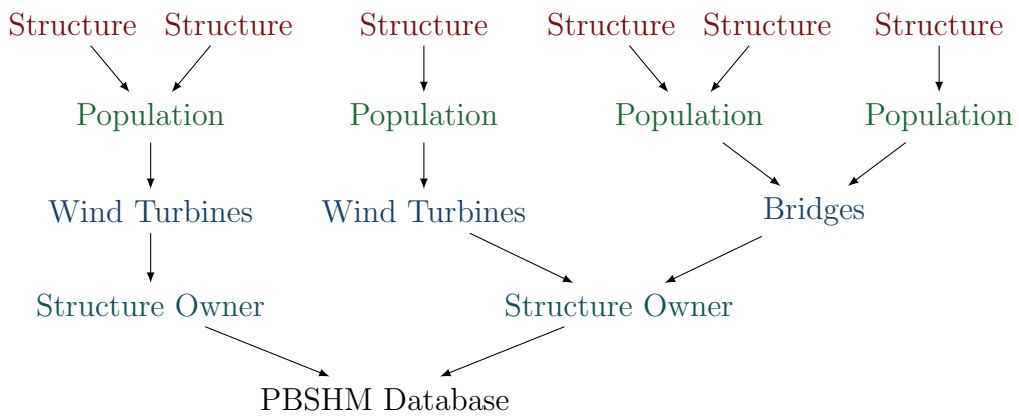


Figure 4.1: Data transmission from structure owners' current SHM systems into a PBSHM database.

Because of the requirement of working in conjunction with existing systems, the PBSHM database should allow structure owners to capture SHM data as their current business practices define. Afterwards, data should be submitted into a centralised PBSHM database. The PBSHM database needs to not only have a standardised format for data storage, but also one for data transmission. The PBSHM database must allow operation in the following scenarios: an open global database that structure owners can submit data into, or a private internal database allowing an owner to compare data across their assets (e.g Wind and Wave farms). Whichever route a structure owner may decide, the format of how to store and communicate data must be the same. (see Figure 4.1)

4.1.1 Database

Selecting the correct database technology is crucial to the adoption of the PBSHM schema and database. Subsequently, each database type is required to be evaluated against the PBSHM requirements (see Section 4.2) set forth in this chapter.

Relational Database Management Systems

Relational Database Management System (RDBMS), in its most simplified version, stores data in the format of tables and columns (see Figure 4.2). Relationships within are created via *primary keys* and *foreign keys*. Primary keys are the method

Structure table

id	name	type
1	structure-1	turbine

Structure Sensor table

id	structure id	sensor name
1	1	sensor-1
2	1	sensor-2
3	1	sensor-3
4	1	sensor-4

Figure 4.2: A relational example showing a "structure-1" structure containing four sensors within an RDBMS.

for identifying a unique data entry inside a table. Foreign keys are the method for creating a link/reference to a column in another table, usually a primary key; thus creating a relationship between the tables. Utilising the combination of primary keys and foreign keys facilitates the retrieval of relational data. To encourage separation of data into multiple tables for the purpose of data deduplication, RDBMS have a normalisation concept.

Database normalisation provides a methodology for when to partition data off into a separate table by comparing the data inside the structure to the normal forms; consequently, it structures data in an approach that grants extensions to the structure, without the necessity of invalidating existing data. Codd [50] introduced what is now known as first normal form (1NF) and later introduced second and third normal form (2NF and 3NF) [79, 80]. Each level of normal form adds additional requirements to reduce data duplication and improve data integrity; the greater the number of columns that contain identical data regarding an entity, the greater the number of columns a system is required to update when data change.

The benefit of an RDBMS is proven system stability; however, they lack the flexibility to enable yet-unknown data to be added into the database without a schema change (see Section 4.2.3 for the definition of a database schema). It is possible to design around this issue in a data schema, nevertheless it is not recommended, because of the required complexity of said schema. In the use case of PBSHM, multiple Structured Query Language (SQL) statements — the standard language used within an RDBMS to interact with the data contained within itself — are required to retrieve the state of a single structure.

NoSQL

NoSQL databases are generally agreed to be divided into four sub-categories;

- Key-Value databases;
- Wide-Column databases;
- Document databases;
- Graph databases.

Key-Value databases are, at their core, a dictionary data structure; they are a large collection of key-value pairs, requiring each key to be unique. Wide-Column databases are similar to RDBMS; they use tables, columns and rows. However, each row inside the database may have differentiating columns compared to sibling rows; similar to a 'dictionary of dictionaries' data structure. Graph databases use vertices and edges to store data; they function via the same rules as graph theory [11, 12]. Document databases use the notion of an entry being a document; they follow a similar structure to *Object Oriented Programming* (OOP). Documents have multiple properties/attributes; however, they can also have nested documents within themselves. Zafar *et al.* [57] discuss the merits and drawbacks of each type of NoSQL database in additional detail.

```
{
    "id":1,
    "name":"structure-1",
    "sensors":[
        "sensor-1","sensor-2","sensor-3","sensor-4"
    ]
}
```

Figure 4.3: An embedded example showing a "structure-1" structure containing four sensors within a NoSQL Document database.

Within the context of a PBSHM database, Document databases permit the functionality of both enabling unknown data to be added into the database without prior knowledge and enabling nesting of data within one document (see Figure 4.3). There is no feature to store relationships between documents; however, in the case of PBSHM databases this is not a problem, given that any relationship between structures would need to be discovered via similarity metrics in the *network* first

(see Section 4.2.1). Document databases provide a unique opportunity for all relevant information regarding the state of a structure to be returned with one document, thus facilitating a straightforward and simple schema for engineers to understand.

4.2 Network, Framework, and Database

In Section 2.1.3 the premise of PBSHM is stated as: *For a given population of structures, determine the similarity between the individual member structures and subsequently, transfer any learnt knowledge across member structures where the aforementioned similarity has been established.* This summary of PBSHM, require two themes addressing; finding which structures (*or components of structures*) are similar, and transferring learnt knowledge across established similar structures.

Whilst the focus of this thesis is on the structural similarity portion of PBSHM, there is a requirement of a shared domain for data residency, in which these structural comparisons and transmission of knowledge can occur. The concept of the Irreducible Element (IE) model introduced by Gosliga *et al.* [6], is reconstructed in Chapter 3 to facilitate a shared domain and language for a standardised description of structures. Tsialiamanis *et al.* [8] address the issue of a shared domain for the transferring of knowledge by proposing the use of a total feature space across a population, whereby mappings can occur from each structure's SHM-driven feature space, using the premise of a fibre bundle throughout established similar structures.

There are further shared-domain problems which require addressing, to enable the technical implementation of these theories within an expandable ecosystem; a shared domain for the knowledge of established relationships between structures (*the network*), a shared domain for the computational algorithms to determine structural comparisons and transmission of knowledge (*the framework*) and a shared domain for standardised data representation and residency (*the database*). By implementing the aforementioned shared domain in a technical ecosystem outlined in this chapter, the focus of PBSHM can shift towards algorithmic discovery and implementation, as, once implemented within the framework, each algorithm will be able to compute against every structure within the network, using the data available within the database.

4.2.1 The network

The *network of structures* – as the name suggests – is the shared domain space in which the relationships between different structures within the PBSHM Database (see Section 4.2.3) reside. Each relationship within the network will have been established via an algorithm implemented in the framework (see Section 4.2.2). The primary vector for establishing these relationships between structures is envisioned to be via varying similarity metrics. Each structure within the network will have relationships of differing weightings to every other structure within the network. One structure may have multiple relationships with a given other structure, dependent upon differing criteria for each available similarity metric.

When a new structure is inserted into the database, the framework will compute similarities between the inserted structure and existing structures. This new structure — and its associated relationships — are then introduced into the network. Populations are then extracted from the network, dependent upon the given strength of relationships present within the network and any observed criteria for transferring knowledge. The known state of a structure is defined by the available data present within the database at the given time.

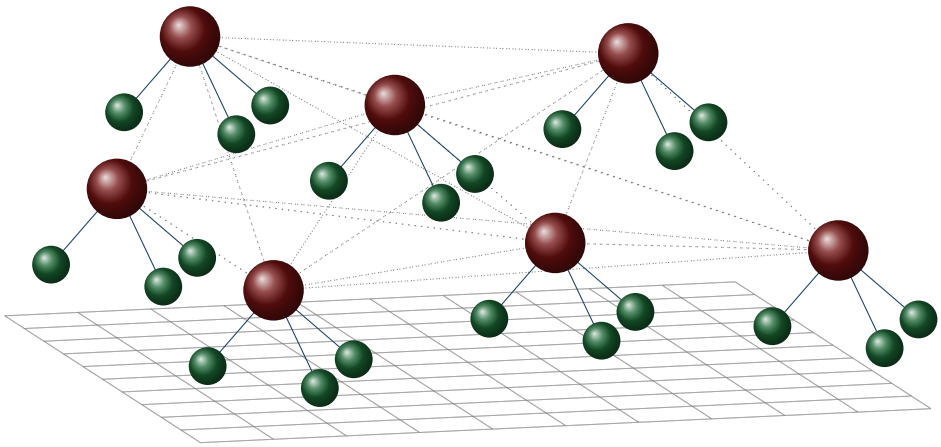


Figure 4.4: A visual representation of the network of structures stored within the PBSHM framework and PBSHM database. The large red nodes represent the presence of a structure within the network, the green nodes represent the data stored within the PBSHM database which depict the current known state of the given structure. The dotted links between the structure nodes represent the relationships present between structures, because of similarities metrics generated by the PBSHM framework.

Figure 4.4 depicts a visual representation of a potential network. If one imagines that each large red node represents an individual structure within the network, each green node with a link to a structure node, represents a piece of data present within the database which establishes the current known state of the given structure. The dotted links between each of the structure nodes, represent a relationship established by the similarity metrics available within the framework. Within the fullness of time, the database is expected to support the full range of data required to capture the state of a structure; however, in the interest of brevity, this chapter will focus on the implementations required to support *IE models* and *channel data*.

4.2.2 The framework

The power of the PBSHM methodology becomes most apparent when the PBSHM ecosystem has a rich spectrum of structures, data, and algorithms present. Additional structures mean increased possibilities of structural similarity and potential opportunity for learnt knowledge to be transferred. However, for this to be achieved, PBSHM is not only going to have to evidence increased knowledge on the health of a structure compared to traditional SHM, but it is going to have to facilitate easy buy-in from key structure stakeholders. There is no point creating world-class SHM algorithms if only a handful of people can interact with the algorithms and understand the outputted information. As such, the framework must not only provide a shared domain in which PBSHM computations must reside, but facilitate a simplistic way in which engineers can interact with both the algorithms and the underlying data.

To support the ever expanding nature of PBSHM, the framework cannot be fixed in nature and must support the organic design principles outlined within the PBSHM database (see Section 4.2.3). In practise, this requirement necessitates an implementation approach which allows compartmentalisation of algorithmic functions into separate units of logic; *a module*. The PBSHM framework will be composed of a central 'core' which deals with the basics of any software platform; authentication, permissions, security, database access, etc. This 'core' will then be expanded into the PBSHM framework by the introduction of modules. Using the implementation pattern outlined above, enables the PBSHM framework to be flexible in its desired functionality and use.

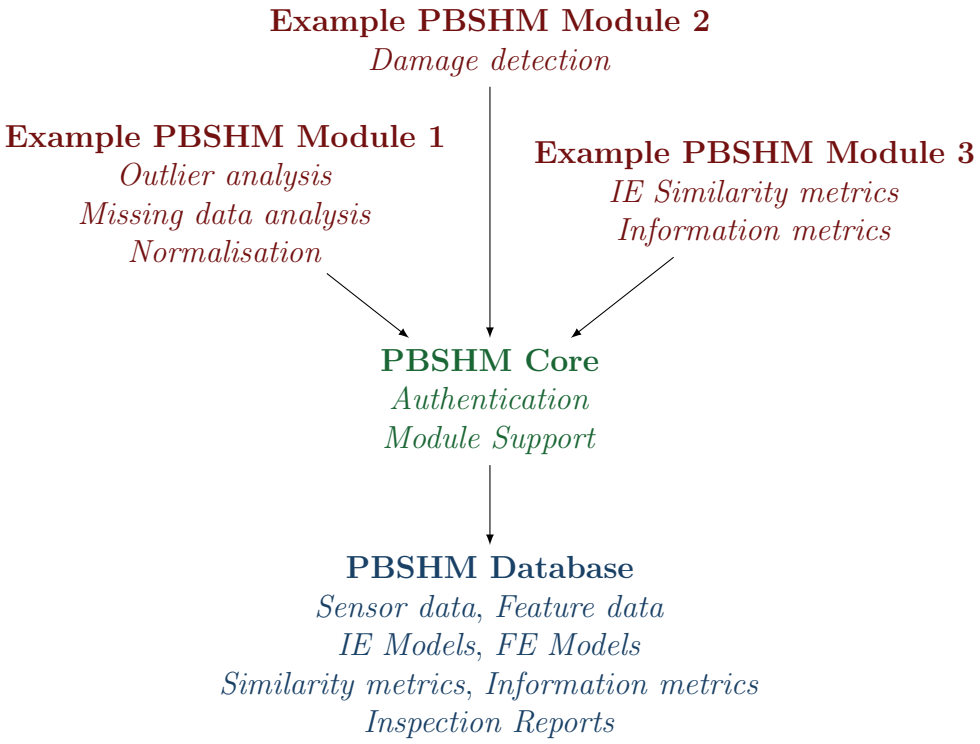


Figure 4.5: The hierarchical structure of the PBSHM framework; PBSHM modules interacting with the PBSHM core and subsequently the PBSHM core then interacting with the PBSHM database to fulfil the requests from the PBSHM modules.

Whilst the ideal scenario for the PBSHM framework may be to have a singular centralised system which encapsulates all the available PBSHM data, the author realises that this may not be practical within the real world and limitations of commercial agreements with structure owners. As such, the flexible nature of the PBSHM framework lends itself to tailoring the installed setup of the framework, dependent upon the structure owners' requirements and available data within the associated database. Figure 4.5 depicts the overall hierarchy of how the modules, core system and database interact together, to fulfil the desired functionality of the PBSHM ecosystem.

During the fullness of time, it is expected that the PBSHM framework (and underlying database), will be expanded via modules to support the full spectrum of data and algorithms required to support the lifespan of a structure; channel data, features, Irreducible Element models, Finite Element models, metadata, inspection reports, etc. However, as the implementation of these future modules

will be decentralised from the author of this thesis, a set of guidelines and requirements are included within, to ensure all modules generated interact seamlessly together and comply with the spirit in which the PBSHM framework is produced.

- Any actions taken upon a population, must function regardless of the population size. Modules must be written in a manner which enables any action or method applied to a population of structures, to also be applied to an individual structure from within that population, if desired. For example, any normalisation of channel data within the PBSHM framework, must function to normalise the data for only one structure or a whole population of structures.

- Raw data, such as data captured during an SHM campaign should be treated as Write Once Read Many (WORM) data. Modules must adhere to the practise of being able to only read raw data from the database and saving any computed data relevant to the module in a different section of the database. The PBSHM framework will enforce a special section of the database where raw data can only be written to by specialised data ingestion modules; however, once the data have been written into the database, changes to this data will not be permitted, regardless of the type of module attempting the change.

- Any discovered features, must have the ability to be saved within the database as either: user-defined features (only available to the user who generated the feature), or global-defined features (accessible by any user within the associated framework). Each feature must be accompanied by a set of metadata describing the details regarding the feature and the relevant framework steps (inputs, algorithms, and variables) taken to generate the feature.

- A module must only use currently-available data from within the framework's associated database. The purpose of a module is to interact with the data contained within the PBSHM database. As mentioned previously, this may be as simple as importing external data of a known format into the database. Alternatively, this also may be as simple as calculating the normalised set of a population's raw channel data and saving it back within the database. A module should not require connecting to a third-party data source to perform its operations on the PBSHM database.

- For a framework to be classed as a PBSHM Framework, there must be at least one module which fulfils the two outlined objectives of PBSHM (see Section 4.2). This requirement means that within a PBSHM Framework, there must be one module which can produce a metric to determine which structures are similar, and another module which can transfer learnt knowledge from one structure to another, given that the metrics in the first module conclude that the structures in question, are from the same 'similar' population.

- Modules should aspire to keep interactions with end users as simple as is relevant for the desired audience of the module. Where possible, algorithmic outputs should be visualised with a detailed explanation of the included visual and any suggestions on what the results of the visualisation may derive.

- Modules may be implemented in whichever language is most relevant for the desired purpose of the module; however, the modules must be able to be run on a Linux distribution and a wrapper must be provided between the chosen language of the 'core' and the implemented module language. For instance, if the developers of a module have chosen the language STAN as the most relevant for their desired purpose and the 'core' has been implemented in Python, the module developers are required to implement a Python wrapper from their module to tie up the interactions between the Python 'core' and the STAN module.

4.2.3 The database

Underpinning the ecosystem described within this chapter, is the PBSHM database. The database fulfils two key roles within the PBSHM ecosystem; providing a shared domain for data residency, and providing the shared standardised format in which PBSHM data is stored (*a schema*). A schema is to a database what an architect's drawing is to a structure; it is a technical document that says where data should go, what is allowed where, and outlines any relationships and hierarchical structure of the data.

Irrespective of the platform chosen to implement the PBSHM database, there are several considerations that must be taken into account in the design of the database – and underlying schema – to ensure that the database fulfil its desired purpose within the PBSHM ecosystem.

- PBSHM will, by its very nature, require vast quantities of varying types of data. Whilst it is not expected that the database will have the knowledge regarding all the types of data required to be stored within, the database must be expandable at a later date to include these yet-unknown data types.

- For a database to be a valid PBSHM database, it must support both of the main themes of PBSHM. As such, the database must have a storage mechanism and shared standard for: describing structures with the outlook of being able to determine the similarity of said structures within the network, and knowledge regarding the state of a structure in which this acquired knowledge could then be transferred across the population.

- Provide and restrict access to certain subsets of data. To satisfy the requirements of external structure owners and the requirements of the framework, the database must provide the facility to restrict access to certain subsets of data. The reasoning behind this is twofold; to protect other entities from gaining access to potentially-sensitive corporate information, and to facilitate the premise of user-defined and globally-defined features within the framework.

- Protect existing data from being overwritten and modified. The framework requirements state that raw data from a monitoring campaign, should be treated as WORM data. The database should adhere to these principals and provide the facility to protect data from being overwritten. One of the founding principals of PBSHM, is that additional knowledge can be gained from utilising multiple sources of data; however, this learnt knowledge must never replace the original source data from which the knowledge was learnt.

- Data must remain valid, once introduced into the database, throughout the lifetime of the data within the system; via the process of expanding the schema to include additional known data types, existing data must remain valid.

- Vast amounts of SHM datasets in question, may be historical and as such, may not have the complete data available that the schema requires. The schema should only enforce the data required for each section of the database, for the data to have valid meaning within its corresponding section.

- Engineers from different fields may be generating and consuming PBSHM data, thus the schema must be straightforward to understand and master.

The schema must not become another barrier for the adoption of PBSHM technologies.

In short, the shared-data domain must be organic in its governing nature. The design of the shared-data domain must include within its principles that there is only a partial understanding of the final data requirements of the domain. This methodology allows for current knowledge to be embedded within the schema at the outset; however, leaving space within the schema for yet-unknown data to be included at a later date. It is imperative for the success of the PBSHM ecosystem, that the database can expand over time with the ever-changing requirements of PBSHM.

4.3 Implementation

Throughout the previous sections of this chapter, the foundations, and principals of a PBSHM ecosystem have been outlined; however, the desire of this chapter is to not only provide the theoretical technical components of such an ecosystem, but to provide an initial implementation of this technology to demonstrate the potential of PBSHM. As such, the rest of this chapter focusses on providing an initial database *schema* for the shared-data domain required within PBSHM and the skeleton of an initial framework for the computational shared domain.

As outlined within the requirements set forth in Section 4.2, any implementation of the ecosystem must support both of the technical themes of PBSHM, therefore, the database and framework implementation outlined within this chapter will focus on supporting the newly reconstructed concept of an IE model from Chapter 3 — to fulfil the requirement of structure descriptions and subsequently similarity comparisons — and raw channel data — to fulfil the acquisition of health knowledge of a structure.

4.3.1 Database choice

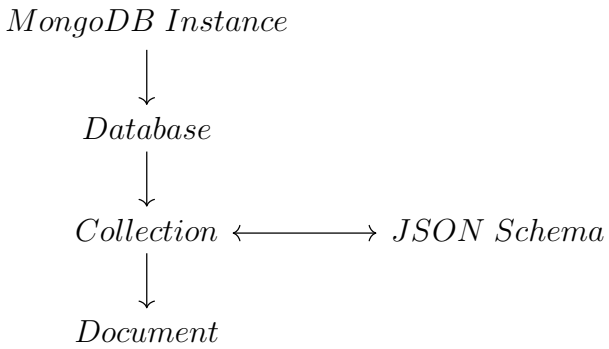
Both RDBMS and NoSQL Document databases provide the technical requirements to implement a PBSHM database. However, with an RDBMS there are additional required processes as a result of converting tabular data into object

representations. Consequently, NoSQL document databases have been chosen here for implementation of the PBSHM database.

There are multiple NoSQL document databases available, each providing a unique set of features driven by community goals. When selecting a document database, the following additional requirements were introduced: access controls, to enable different structure permissions, simple document structure and document level concurrency. For the purposes of the PBSHM database, MongoDB has been chosen. MongoDB was initially developed in 2007 [81], and has since become a widely-adopted and supported document database. MongoDB can be deployed on premises, on cloud providers or via MongoDB Atlas.

Documents inside MongoDB are stored in Binary Javascript Object Notation (BSON) format - a derivative of the Javascript Object Notation (JSON) format. JSON [82, 83] is a text-based data format that has gained popularity precisely because of its transparent document structure, lightweight footprint and self-contained nature.

MongoDB uses the following levels for data storage:



An instance of MongoDB can contain multiple databases; inside each database can be multiple collections and inside a collection can be multiple documents. Whilst a principle of NoSQL document databases is a schema-less design, MongoDB supports document validation via associating each collection with a JSON Schema. This facilitates a “pbshm” database inside the MongoDB instance, a global “structures” collection inside the database using the PBSHM JSON Schema for document validation, and PBSHM data documents inside the collection.

Because of MongoDB’s access controls [59, 84], all users will be given read permission to the “structures” collection; if any PBSHM framework modules require storage of

computed values, these will be stored in module-specific collections. The MongoDB database here will use the WiredTiger [59, 85, 86] storage engine, which supports document-level concurrency, thus facilitating a structure's data to be updated on the system without locking read access to the rest of the structures.

Minimal latency upon retrieval of common queried data is implemented via *indexes* [59, 87]. Indexes are quintessentially pointers from a specific data value to documents which are stored in memory. For example, an index could be created on a population name, so that when the system is queried for documents inside a population, the system already knows which corresponding documents to be return without enumerating each document inside a collection.

Distribution of data, either for redundancy or scalability is a key component for any database. MongoDB supports this via *replication* [59, 88] and *sharding* [59, 89]; however, for the purpose of this chapter, they are not implemented within the PBSHM database, as the data used for the documentation can fit within a single database node.

Structure

At the root of a PBSHM data document, is knowledge regarding the 'who' and 'when' of the shared data domain; *the structure*. The basic properties required to capture this are: *version*, *name*, *population*, and *timestamp*. Additional properties can be included within the Structure entity to encapsulate data from within the PBSHM framework. In the examples used within this chapter, additional properties have been declared for models and raw channel data; *model* → *irreducibleElement* (see Chapter 3), and *channels* (see Section 4.3.1) respectively. Figure 4.6 depicts the hierarchical objects for the *structure* object and Table A.1 gives a full list of all the properties, including which properties are required.

Within MongoDB there is a default *date* property type which stores date information up to millisecond accuracy; however, because current sensor technology can capture data at a 1Mhz sampling rate, this would cause a loss of accuracy on sensor data. Dates within the PBSHM database are implemented via UTC nanoseconds since UNIX epoch and stored in a *long* (Int64) data type which enables sampling up to 1Ghz[1].

[1]When using Javascript to interact with the PBSHM Schema, there will be a loss of accuracy

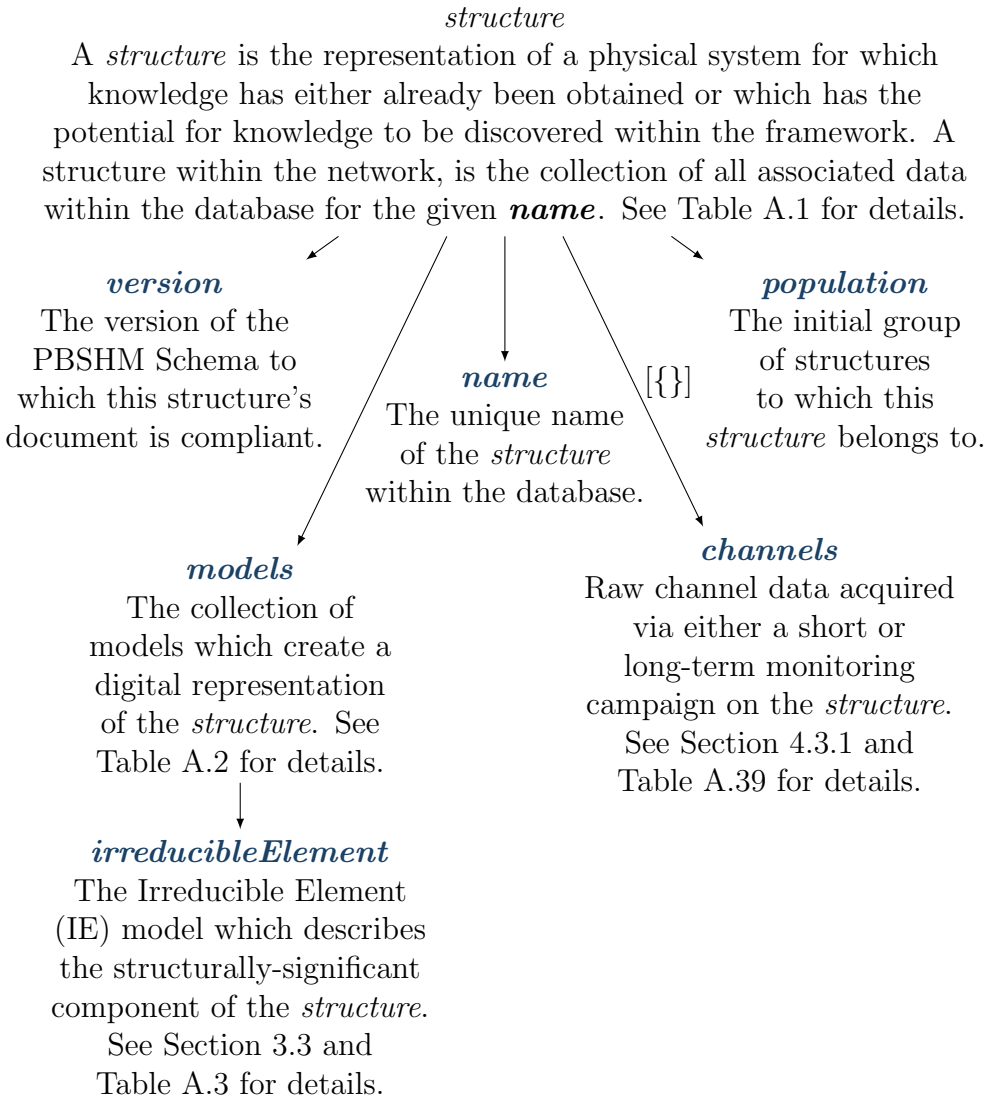


Figure 4.6: The hierarchical layout of properties and child objects within the *structure* root document.

Channel

Embedded beneath the Structure document is an array of Channel objects containing details on associated Structure sensors; *name*, *type*, *unit* and *value*. A Channel object should exist within the Structure document for each Channel that was present and providing data on the associated Structure at the given Structure timestamp. If a Channel did not provide data at the given timestamp, it should not

in the timestamps because of Javascript only supports 53 bytes in an Int64. This constraint will also affect any interactions within the MongoDB Compass GUI tool, as at the time of publishing this chapter it is implemented in Javascript

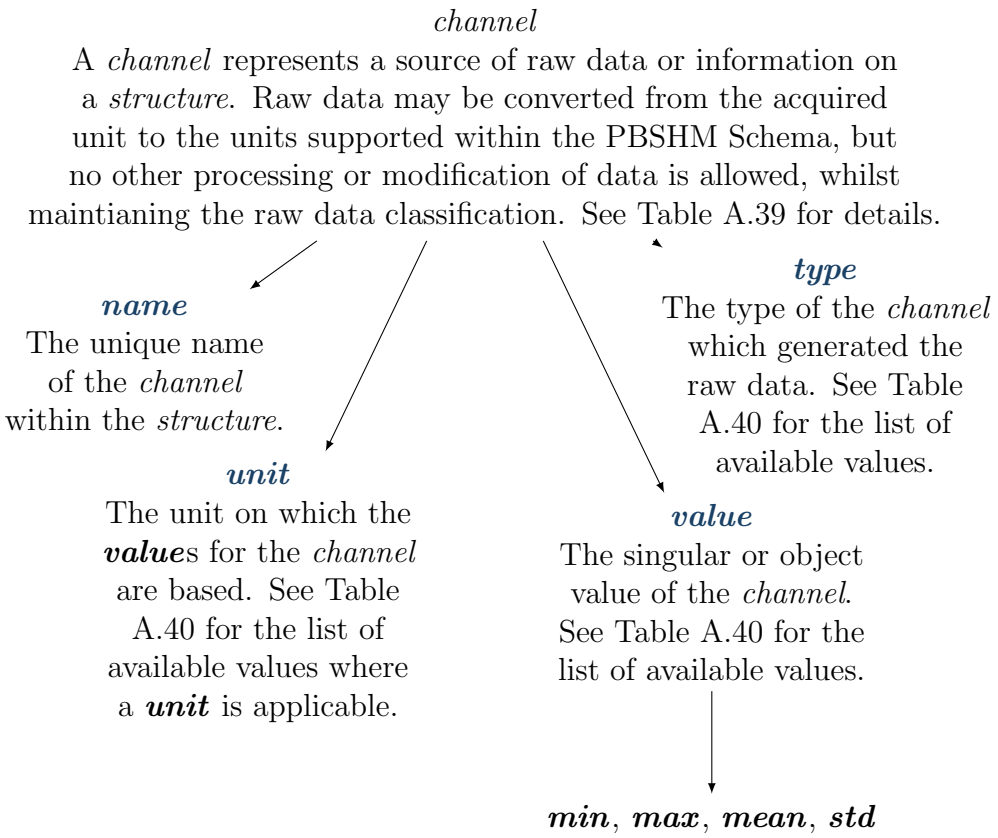


Figure 4.7: The properties for a *channel* in the PBHSM Schema.

exist within this Structure document. Figure 4.7 depicts the hierarchical objects for the *channel* object. A description of each property within the Channel entity is included in Table A.39.

To enable accurate comparisons across Channel objects, ***type*** and ***unit*** are enumerated types. For each given value of ***type***, there are associated accepted values for both ***unit*** and ***value***. For instance, if a channel has a value 'tilt' for ***type***, accepted values for ***unit*** are 'degrees', 'radians', or 'other' and accepted types for ***value*** are *int*, *double* or *object*. The ***unit*** property is not applicable on certain *type*s where having a *unit* to provide context for the ***value***s value isn't required. For example a Channel with ***type*** 'text' does not require any additional context for the value of ***value***. A full list of accepted ***type***, ***unit***, and ***value*** combinations is included in Table A.40.

```
{
    "version": "1.1.0",
    "name": "test-structure",
    "population": "ie-population",
    "timestamp": 1588007841000000000,
    "models": {
        "irreducibleElement": {
            "type": "grounded",
            "elements": [
                {
                    "name": "ground-element",
                    "type": "ground"
                },
                {
                    "name": "regular-element",
                    "type": "regular",
                    "contextual": {
                        "type": "beam"
                    },
                    "geometry": {
                        "type": {
                            "name": "beam",
                            "type": {
                                "name": "rectangular"
                            }
                        }
                    },
                    "material": {
                        "type": {
                            "name": "metal"
                        }
                    }
                }
            ],
            "relationships": [
                {
                    "name": "relationship",
                    "type": "boundary",
                    "elements": [
                        { "name": "ground-element" },
                        { "name": "regular-element" }
                    ]
                }
            ]
        }
    }
}
```

Figure 4.8: An JSON example for including IE model data within the PBSHM database. The example is a [grounded] model of a simple rectangular beam interacting with *ground*.

Value

When the ***unit*** for a Channel is applicable, the ***value*** on the Channel can be of data type *int*, *double* or *object*. A Value object facilitates storage of *value* data which is not singular; ***min***, ***max***, ***mean***, ***std***. The Value object must have at least two properties set on the object, otherwise a singular ***value*** value should be used instead. A description of each property within the Value entity is included in Table A.41.

4.3.2 JSON examples

To illustrate the usage of the PBSHM Schema, the following JSON examples have been included to demonstrate the relationship between the Schema outlined within this chapter and the JSON code required to represent data into the database. Figure 4.8 includes the JSON required to represent a [grounded] *model* within the database. The example is of a simple rectangular beam, represented by a singular [regular] *element*. The model contains the minimum required information for the model to be valid and pass the Schema validation.

Figure 4.9 includes the JSON required to represent a structure with three channels of data for the given timestamp. The first channel 'test-channel-1' is a SCADA channel with the source being an acceleration sensor. The second channel, 'test-channel-2' is a singular value channel with the source being a temperature sensor. The last channel, 'test-channel-3', is a text-based channel of non-numerical data. A single timestamp document may contain channels of varying channel types and values. Whilst an example is not given within this chapter in the interest of brevity, a single JSON document could contain both IE model data and channel data.

4.3.3 Framework

In the interest of conciseness, the complete implementation of the framework is omitted from this chapter; however, an overview of details regarding the implementation have been included to show the research decisions made and the main technology details. Given the requirements of the framework outlined in Section 4.2.2, a Python-based micro web framework called Flask has been chosen as

the base technical implementation upon which the framework is built. A PBSHM flask core has been implemented to provide a skeleton structure in which modules can be created to expand the functionality of the 'core' to produce a flexible framework.

Keeping in mind the requirements outlined within this chapter of what is required for a PBHSM framework, two modules have initially been authored to fulfil the aforementioned requirements. Figure 4.10 shows a module within the PBSHM framework which is an implementation of the maximum common subgraph and the Jaccard Index originally used within Gosliga *et al.* [38] paper to determine the similarity between a set of eight bridges. The IE models of the associated bridges,

```
{
    "version": "1.1.0",
    "name": "test-structure",
    "population": "test-population",
    "timestamp": 1588007839000000000,
    "channels": [
        {
            "name": "test-channel-1",
            "type": "acceleration",
            "unit": "g",
            "value": {
                "min": 10,
                "max": 11,
                "mean": 10.5,
                "std": 10
            }
        },
        {
            "name": "test-channel-2",
            "type": "temperature",
            "unit": "C",
            "value": 28.9
        },
        {
            "name": "test-channel-3",
            "type": "text",
            "value": "this is some text based context"
        }
    ]
}
```

Figure 4.9: An JSON example for including channel data within the PBSHM database. The example includes three channels, the first channel demonstrates SCADA data within the database, the second channel a singular value and the third channel with only a text-based value.

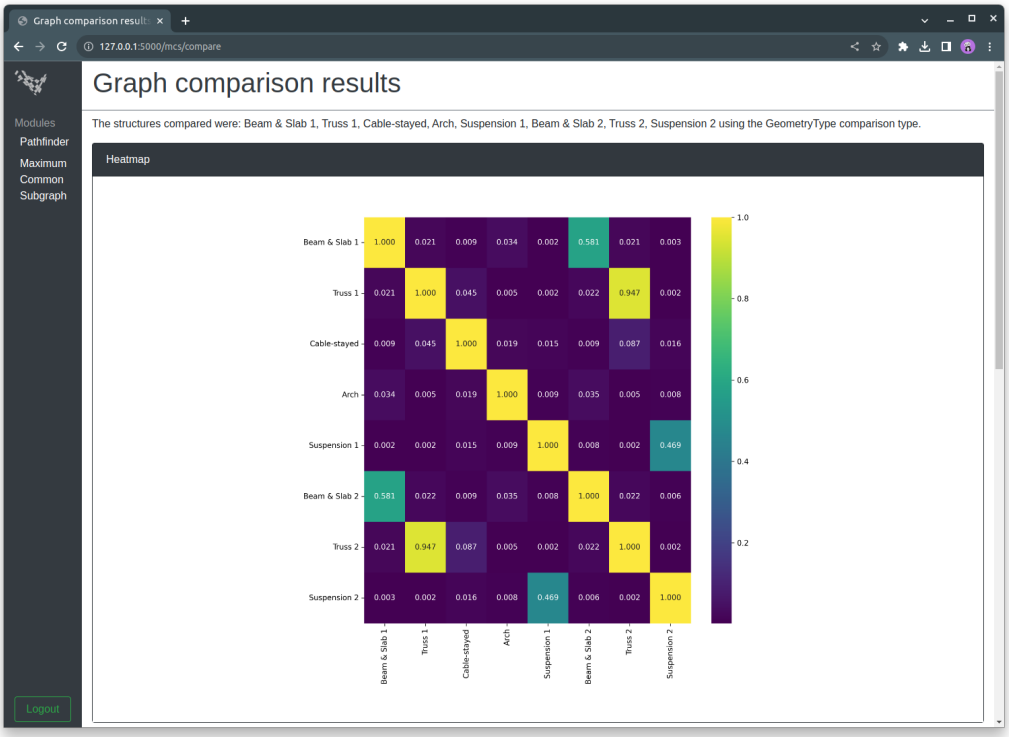


Figure 4.10: An output from the PBSHM framework on computing the similarity of eight bridge structures using the Jaccard Index. The bridges included within the comparison are the same set of bridges used by Gosliga *et al.* [38].

have been updated to the latest version of the IE model definitions included within this chapter. Figure 4.11 shows the 'pathfinder' module within the framework which enables exploration of existing datasets within the database and includes basic statistical analysis of these datasets.

4.3.4 Download

The schema and framework introduced in this chapter can be downloaded from `https://github.com/dynamics-research-group/pbshm-schema/releases/tag/v1.1.0` and `https://github.com/dynamics-research-group/pbshm-flask-core/releases/tag/v1.0.4` respectively. For manual installation within MongoDB, the required file from the schema repository is *structure-data-compiled-mongodb.min.json*. To configure the framework, please follow the instructions in the *README* file on the framework repository. During the configuration of the framework, the latest version of the PBSHM schema will be downloaded and

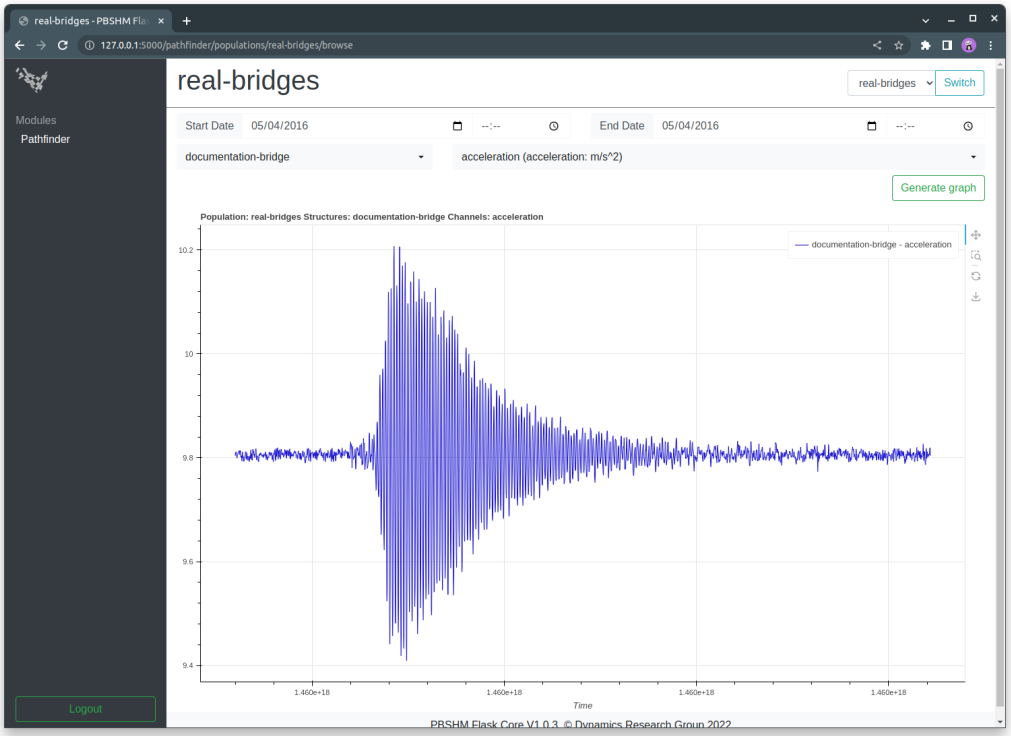


Figure 4.11: An output from the PBSHM framework on browsing the available *channel* data for a given population.

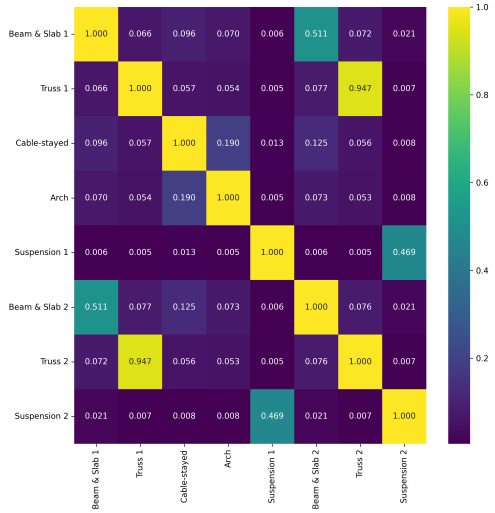
installed within the selected *collection*.

4.4 Conclusion

In conclusion, this chapter has highlighted some significant data problems that could potentially plague PBSHM, if not dealt with whilst PBSHM is in its infancy. The chapter has proposed the potential solution for this, as a shared domain in which PBSHM data and algorithms can reside, to not only aid in the adoption of PBSHM, but enable the focus of PBHSM henceforth to be on algorithmic discovery and generation, instead of data headaches.

This chapter built upon the grounds of Chapter 3 and has introduced the idea of three unique shared domains; the *network*, the *framework*, and the *database*. The *network* houses the similarity computations between structures, and creates a network of relationships between the structures within PBSHM to evaluate if learnt knowledge can be transferred across the *network*. The *framework* is where PBSHM

(a) The framework's similarity matrix from the eight bridges, when embedding the geometrical type data into the Attributed Graph.



(b) The framework's similarity matrix from the eight bridges, when embedding the contextual type data into the Attributed Graph.

Figure 4.12: The similarity matrix generated from the PBSHM framework using the Maximum Common Subgraph and the Jaccard Index. The Irreducible Element models used are the same eight bridges used by Gosliga *et al.* [38].

algorithms reside; any computations on the similarity of structures or determining potential transfer of knowledge from structures, happen here. The *database* unifies both the *network* and *framework* by providing the shared-data format against which all PBSHM data must be compliant.

The importance of the changes outlined within Chapter 3 to the language and knowledge embedding within an IE model become apparent when one examines the output from this Chapter's framework's similarity metrics for the eight bridges used by Gosliga *et al.* [38]. Figure 4.12a shows the output from the framework when only embedding the geometrical type properties from the IE model into the Attributed Graph. As one can see, the similarity metrics still correctly identify 'Beam & Slab 1' and 'Beam & Slab 2' as being similar, the same is for 'Suspension 1' and 'Suspension 2', and for 'Truss 1' and 'Truss 2'. These results are all in line with the result published by Gosliga *et al.* However, one may notice that the 'Cable-stayed' bridge and the 'Arch' bridge no longer have any meaningful similarity between themselves, in contrast to the results published by Gosliga *et al.* Figure 4.12b shows the output from the framework for the same eight bridges as used in Figure 4.12a; however,

the embedding within the Attributed Graph is changed to the contextual type from the IE model. As one can see, the vague similarity between the 'Cable-stayed' and 'Arch' bridges has returned to the same value as published by Gosliga *et al.* This loss and regain of similarity between the 'Cable-stayed' and 'Arch' bridges, highlights the importance of the work included within this chapter and Chapter 3 to isolate knowledge within an IE model to its corresponding knowledge domains.

Chapter 5

Similarity Metrics

The work in this thesis so far has focussed on curating the required practical building blocks to scale similarity comparisons from the initial notion phase to a practical and standardised solution, to enable widespread adoption of PBSHM technologies within the SHM research community. This chapter focusses on the problems faced when multiple people are involved in the curation of models. The ideal situation is that for a single structure, no matter who generates the model of the structure, one should always end up with the same model at the end of the process. Unfortunately, this is inevitably not the case. Each person who generates a model — henceforth know as the author — will have differing desires behind why they are generating the model in the first place — *author bias*. This unavoidably leads to variations being present within the model.

The chapter explores the effects these variations have upon the similarity-comparisons computed with the *network*. In Section 5.1 a background is given as to why these aforementioned variations may appear within the model because of author bias. The *Canonical Form* and the *Canonical Form Reduction Rules* are introduced in Section 5.2 as the potential vehicle within PBSHM to remove these variations from the *network*. Finally, a Graph Matching Network is utilised in Section 5.3 as an alternative similarity metric to determine if a machine learning approach can learn the similarity, even with the variations present within the *network*.

5.1 Background

Within PBSHM, IE models (see Chapter 3) are the vehicle used to embed structural knowledge into the PBSHM; however, they are not the final domain in which this structural knowledge resides. The whole purpose of embedding structural knowledge within PBHSM — and thus the necessity of IE models — is to facilitate the comparison of structures, to collect a measured score of similarity between structures for determining potentially unknown *populations*. The introduced ecosystem in Chapter 4 refers to this final destination of structure as the *network*; a shared domain in which the similarity comparisons of PBSHM structures reside and — based upon the associated similarity — establish the strength of relationships between these structures. The implementation of these similarity algorithms will be present within the *framework* portion of the ecosystem, and as such, may support multiple different similarity algorithms that execute within the *network*. Each structure within the *network* will have a similarity score to every other structure, potentially for each supported similarity algorithm within the *framework*.

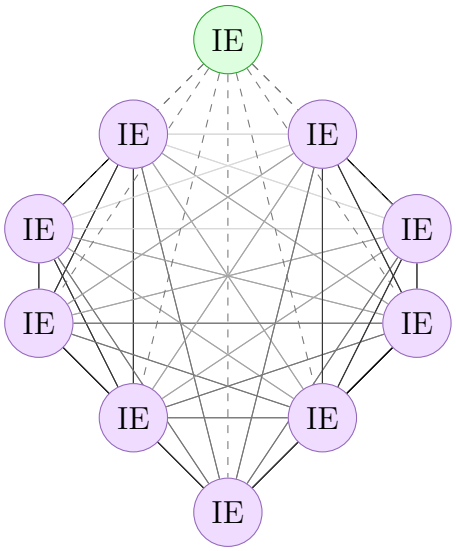
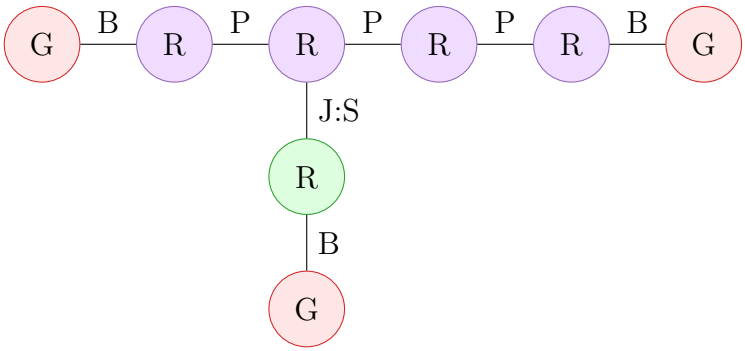


Figure 5.1: A diagram of the similarity score-driven relationships between Irreducible Element (IE) models within the PBSHM network. Each existing IE model — a purple node — within the network, has a relationship with every other IE model within the network — derived from a similarity score. The diagram also depicts a new IE model — the green node — being added to the network, and the process of relationships being discovered between the newly inserted IE model and existing network models.

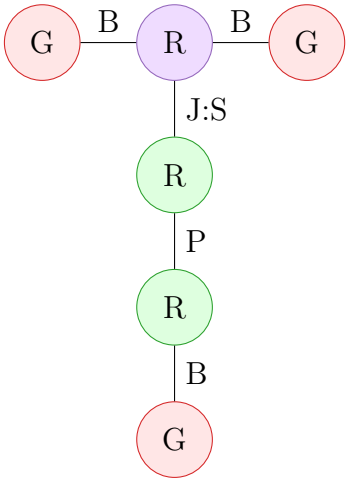
This affiliation of relationships between structures within PBSHM can be envisioned as a complete weighted graph, where each node is the model of a structure and each link is the similarity value between the two structures. Figure 5.1 visualises the relationships between structures within the *network*. As the *network* is the final domain for IE model data, it is only natural that the field of graph theory [11, 12] be an avenue for exploration in the goal of determining the similarity of structures. IE models by their definition, naturally lend themselves to be represented as an Attributed Graph (AG): each *element* becomes a vertex, and each *relationship* becomes an edge. All the knowledge present within the IE model, is then embedded as attributes on the corresponding vertex or edge.

Figure 5.2: A two-span beam-and-slab bridge example from Northern Ireland. The same bridge that is showcased in Chapter 3.

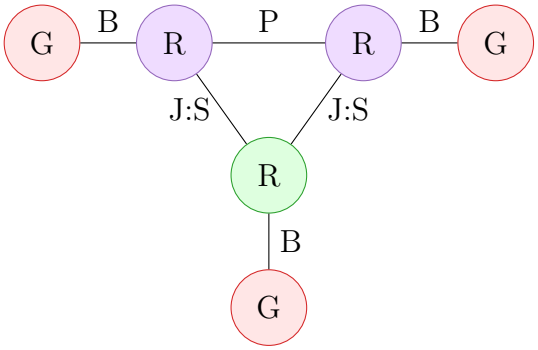
Whilst PBSHM is a relatively recent branch of SHM, it does not invalidate the fundamentals upon which SHM was built, and must honour these principles and practises within the theory of PBSHM. One of these aforementioned principles within SHM, is the desire to locate where potential damage is located within the structure. The issue with honouring this principle is *subjectivity*. If one considers the two-span beam-and-slab bridge depicted in Figure 5.2. The bridge — for the purpose of this chapter — can be simplified into a single beam which runs horizontally from the left embankment to the right embankment — as pictured — and a single column supporting the horizontal beam, from the centre of the beam to the road. During the process of curating an IE model for the given structure, one engineer may be particularly interested in locating the damage on the beam, and as such, add more details within the model on the beam section of the IE model. Another engineer may decide that the damage on the column is of paramount importance, and thus add additional details to the column section of the IE model. These nuances in model objectives may appear insignificant within the grand scheme of PBSHM; however, they vastly change the arrangement of an IE model and thus the associated AG.



(a) This IE model variation has been modelled with the deck as two separate [regular] *elements* with the column component only interacting with the right section of the deck. The right deck has also been subdivided into three [regular] *element*s to aid in damage localisation on that section of the deck.



(b) This IE model variation has been modelled with the deck as only a single [regular] *element*, whilst the column component has been divided into two [regular] *element*s for damage localisation within the column.



(c) This IE model variation has the deck split into two [regular] *element*s, with the column section being modelled as a single [regular] *element*; however, the interaction between the column and the deck has been modelled as interacting with both sections of the deck.

Figure 5.3: Three of the potential Irreducible Element (IE) model representations — displayed as graphs — of the two-span bridge displayed in Figure. 5.2. [ground] *element*s are represented by a G in the centre of the node, [regular] *element*s are represented by an R in the centre of the node. [boundary] *relationship*s are represented by a B on the edge, [perfect] *relationship*s are represented by a P on the edge and a [joint] *relationship* with a [static] nature is represented by a J:S on the edge.

Figure 5.3 illustrates how the subjectivity of the model creator — the *author bias* — can change the underlying model submitted into the *database* and ultimately the *network*. The first graph (see Figure 5.3a), shows the changes present within the IE model if the author decided that instead of the horizontal beam being a single [regular] *element*, the horizontal beam is initially split into two [regular] *element*s to locate damage to a particular span of the bridge, the right span of the bridge is further subdivided into three [regular] *element*s for either sensor placement or potential further damage localisation given signs of wear on that span of the bridge. The second graph (see Figure 5.3b), shows that the horizontal beam has been left as a single [regular] *element*; however, the vertical column has been split into two [regular] *element*s, to enable damage localisation to either the top section of the column or the bottom section of the column. The third and final graph (see Figure 5.3c), splits the horizontal beam into two [regular] *element*s and a single [regular] *element* for the column; however, the engineer generating this IE model has determined that there should be a [joint] *relationship* to either span of the horizontal beam. These are only three of potentially limitless variations that can be present in the simplified two-span beam-and-slab bridge.

Variations being present within a model because of author subjectivity, are a fundamental issue with any modelling task. The problem was present in the initial version of IE models by Gosliga *et al.* [6] and remains present in the newly reconstructed version in Chapter 3; however, with the second version of the IE model language, there is embedded knowledge stored within the model itself to help understand and interpolate why an author has chosen to dissect the structure in the manner present within the model. Research has already been initiated by Gosliga *et al.* [6] into the viability of the Jaccard Index as a similarity metric within PBSHM. The Jaccard Index works by calculating the Maximum Common Subgraph (MCS) between two graphs, in the case here, two attributed graphs.

To evaluate the impact these aforementioned variations have upon PBSHM's similarity results, a synthetic dataset was generated based upon the simplified two-span beam-and-slab bridge example illustrated within this chapter. The dataset contains randomly-generated beam-and-slab bridges from two to ten spans, with each span being potentially divided up into three subsections; furthermore, each column between the span was either joined to the previous span, the next span, or both spans to include the full set of variations presented in Figure 5.3. The dataset contains a total of 4500 randomly-generated bridges — 500 bridges per number

of span — and was then randomly separated into a training, validation and test subsets. This dataset — for the purposes of this chapter — will be henceforth known as, the *matching* dataset.

To ensure consistency throughout the similarity matrix results depicted within this chapter, the embedding of attributes into the AG representation from an IE model has been fixed to embedding only the *contextual* type — the ***type*** attribute value from the *contextual* object within a [regular] *element*. For vertices where there is no *contextual* type — such as a [ground] *element* — no attributes are embedded into the vertex. The edges in the AG representations have no attributes from the associated [relationship]s embedded within the graph.

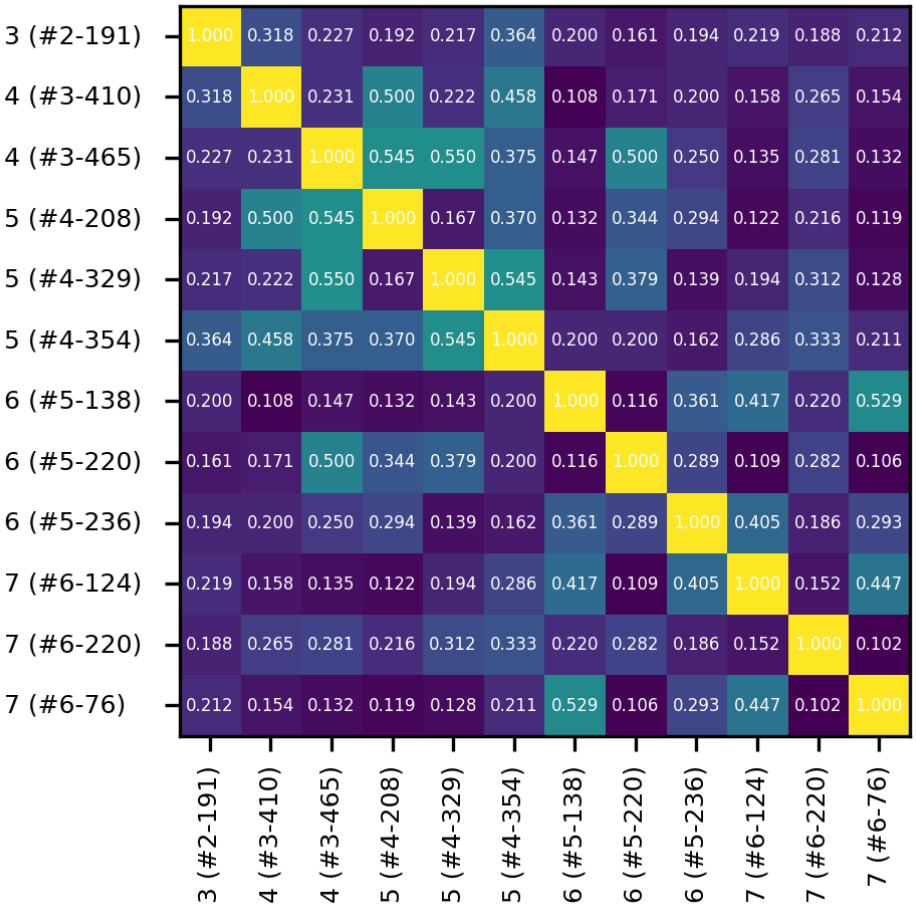


Figure 5.4: The Jaccard Index similarity matrix results from the Maximum Common Subgraph on the test portion of the matching dataset when embedding only the contextual type as the node attribute. The axes are labelled with the number of spans the graph is associated with and the ID of the graph from within the dataset.

Figure 5.4 shows the results of embedding only the [regular] *element*'s *contextual* type within the AG and evaluating each pair within the *network* for their given MCS similarity using the Jaccard Index. The axes of the similarity matrix are labelled with the number of spans of the bridge and their associated graph number within the matching dataset. In the ideal scenario, all the graphs with the same number of spans should all identify as matching with a similarity value of 1. When a graph with either a descending or ascending number of spans — $N-1$, $N+1$ — is compared to a graph with N number of spans, the similarity score should identify these as the next closest match, after N.

As the reader can see in the results in Figure 5.4, when the inherent ambiguity of model author subjectivity is included within the graphs, the algorithm is not able to find any strong recognisable pattern. The algorithm correctly identifies when the graph is compared to itself; however, the algorithm — at least within the matching dataset — is not able to correctly identify graphs with the same number of spans as identical; instead, it identifies graphs with differing number of spans as being the closest matches. If one looks at the result for 6 (#5-220), the algorithm identifies a four span bridge (#3-465) as having a closer similarity than any of the six span bridges.

5.2 Canonical Form

The observed variations present within the similarity metrics — when introducing the inherent model subjectivity — highlights two new scenarios which require attention within the comparison portion of PBSHM. When generating similarity scores, two graphs must always match as identical, if the source structure from which both graphs have been generated, is the same structure, and structures which are classed as nominally-identical or from a homogeneous population [5], should further match as identical.

This chapter proposes that the solution for addressing the aforementioned scenarios across all current and future similarity algorithms within the *framework*, is a methodology for reducing IE models to a common form. A form which preserves the structural knowledge and engineering decisions present within the original model, but facilitates a common representation of a single structure, regardless of any author subjectivity; a *Canonical Form*. IE models generated by authors would

henceforth be known as *detailed* IE models, and only reduced to a *Canonical Form* representation for the purpose of similarity matching within the *network*. *Detailed* IE models would still be submitted by authors into the *framework* and ultimately stored within the *database*.

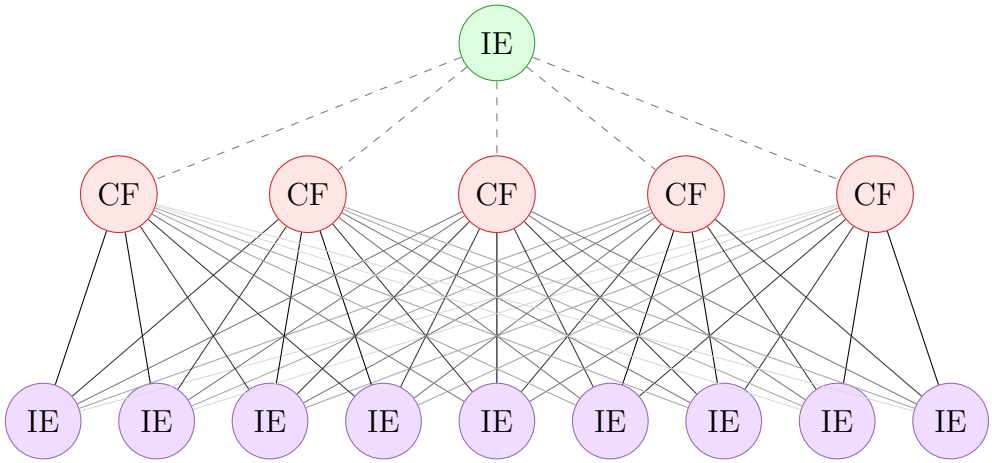


Figure 5.5: The PBSHM network using the Canonical Form as a common form for comparison. The red nodes represent the known *Canonical Form* representations within the *network*. The purple nodes represent existing *detailed* IE models, for whom similarity comparison values are already present against the known *Canonical Form* representations. The weight of the similarity between the existing *detailed* IE models and the known *Canonical Form* representations, are represented by increased darkness of colour on the link — higher similarity scores equal darker links. The green node represents a new *detailed* IE model being inserted into the *network* and the dotted links represent the similarity calculations made upon insertion.

Furthermore, the notion of a common form for a single structure has the potential to improve the performance of the *network*. Currently, the *network* acts as a complete weighted graph for each similarity algorithm within the *framework*. Computationally, this involves each unique pair of graphs having their similarity computed. Whilst this mechanism may appear trivial when factoring a toy dataset numbering only a few hundred graphs, the logistics of performing this same computation become problematic when considering the potentially vast size and quantity of real-world structure graphs. The largest single graph within the matching dataset, has in the order of tens of *element*s/nodes, real-world structures may have *element*s numbering in the order of hundreds or even thousands within a single structure. If one factors in, that within a *network*, one may have thousands, if not tens of thousands of structures present, the reality of performing these computations becomes expensive — without factoring in the possible variations

from model subjectivity.

This chapter proposes that the solution to the computational problem is, that the *Canonical Form* becomes an intermediary layer within the *network*, to act as a known target for comparison against *detailed* IE models. Each *detailed* IE model within the *network*, would have a similarity score to every *Canonical Form* within the *network*. When a new *detailed* IE model is inserted into the *network*, only similarity scores are drawn up for the newly-inserted model and the existing known *Canonical Form*s. The proposed modified methodology of the *network*, has the potential to not only reduce the number of computations performed within the *network*, but also create a natural alignment of populations within the *network* for discovery by clustering algorithms. Figure 5.5 visualises the configuration of the *Canonical Form*-inspired *network*, and depicts the process of a new *detailed* IE model being included into the *network*.

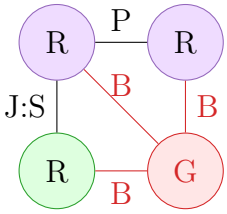
5.2.1 Canonical form reduction rules

To facilitate the process of reducing a *detailed* IE model to the corresponding *Canonical Form* representation, this chapter proposes an initial set of three reduction rules to accomplish the desired common form; the *Canonical Form Reduction Rules* (CFRR). The CFRR are a set of rules which can be applied to any *detailed* IE model, with the goal of removing any ambiguity from the model; however, the rules must not remove any embedded knowledge within the model which may later be used within the similarity metrics. Each rule must be grounded in a solid reasoning as to why the associated modifications contained within the rule are able to modify the IE model, without the loss of knowledge from within the model. While the reduction rules may be applied whilst the model is within the IE model domain, the reality is that the *Canonical Form* representation occurs whilst within the *network* and as such, further discussions within this chapter will refer to the changes made by the CFRR as within the associated graph-domain of the *network*.

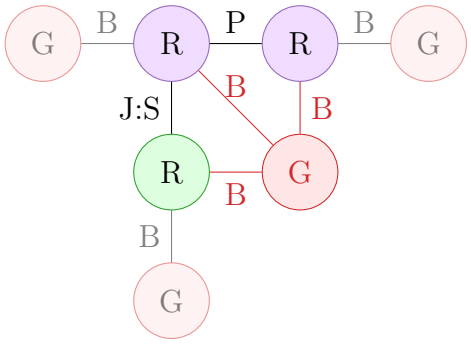
Individual Ground

The first rule proposed within the CFRR, is that each [ground] *element* within a graph, must be unique. This rule requires that wherever a [boundary] *relationship*

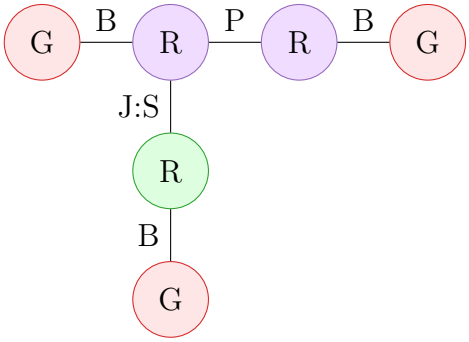
is present within a graph, the associated [ground] *element* included within the *relationship* must be unique to the [boundary] *relationship* and not shared with any other [boundary] *relationship*s. The reasoning behind this rule is, each [ground] *element* present within the graph, is the representation of another structure's presence within the model. Each interaction between the structure being modelled, and the third-party structure is unique, and as such, should be represented as



(a) Step 1: Select any [ground] *element* where two or more [boundary] *relationship*s are present. The pattern is highlighted in red.



(b) Step 2: Create a new [ground] *element* and [boundary] *relationship* for each [boundary] *relationship* present within the previously selected [ground] *element*. The creation of new *object*s, must retain within the target *object* (i.e. the newly created *object*) any attributes present within the source *object* (i.e. the *object* which is being copied).



(c) Step 3: Remove the previously-selected [ground] *element* from the model.

Figure 5.6: The stages of a Individual Ground Canonical Form reduction against an Irreducible Element model graph. By performing this reduction, an unrequired loop is removed from the graph without the loss of any embedded knowledge within the model.

a unique [ground] *element* within the model. As a [ground] *element* is only the reference to the presence of an external structure, no knowledge is lost by this reduction rule.

The *Individual Ground* reduction rule, not only reduces the topological complexity of the graph by removing unnecessary loops, but could also be applied as a general rule for [ground] *element*s in the *detailed* type. Figure 5.6 illustrates the process of selecting a [ground] *element* with more than one corresponding [boundary] *relationship*, creating new [ground] *element*s and [boundary] *relationship*s, and subsequently, removing the offending [ground] *element* and [boundary] *relationship*s.

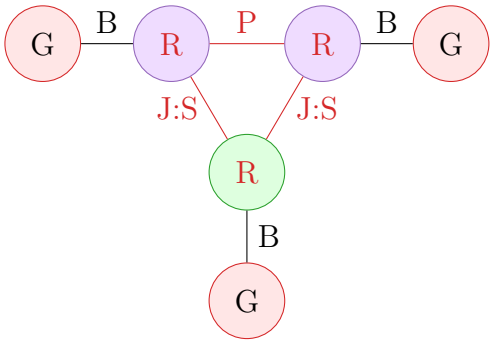
Perfect Joint Joint Relationships

The second rule proposed within the CFRR, is that any time within the graph where there is a pattern of three [regular] *element*s connected in a loop via a [perfect], [joint], and [joint] *relationship*; the loop can be broken and reduced to a [perfect] and [joint] *relationship*. If one takes the example illustrated in Figure 5.3c, there are two [regular] *element*s — representing the horizontal beam in the example bridge — connected via a [perfect] *relationship*, there is then a single [regular] *element* — representing the vertical support column — connected to both of the aforementioned [regular] *element*s of the horizontal beam, via independent [joint] *relationship*s.

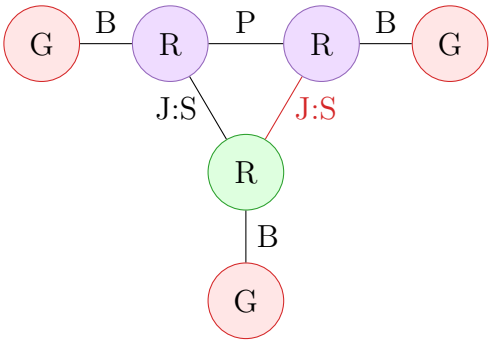
The interaction between the three aforementioned [regular] *element*s can be modelled in three distinct manners: the vertical support column is connected via a [joint] *relationship* to both of the horizontal beam [regular] *elements* (as depicted within Figure 5.3c), the vertical support column is connected via a [joint] *relationship* to only the left horizontal beam [regular] *element*, and oppositely, the vertical support column is connected via a [joint] *relationship* to only the right horizontal beam [regular] *element*.

Each of these scenarios is a valid method for embedding the structural knowledge of the interaction between the horizontal beam and the vertical support column. In the last two scenarios, the physics between the vertical support column and the horizontal beam have been embedded once within the model; conversely, in the first scenario, the physics have been embedded twice within the model, once to each beam.

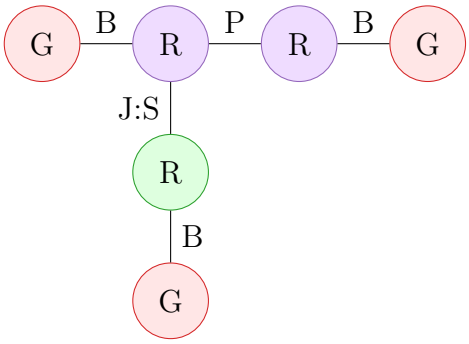
The *Perfect Joint Joint* reduction rule can safely reduce a [perfect], [joint], [joint]



(a) Step 1: Select any loop within the graph, constituted of three [regular] *element*s, connected via a [perfect], [joint] and [joint] *relationship*. The loop pattern is highlighted in red.



(b) Step 2: Select one of the [joint] *relationship*s within the loop. The selection between which of the [joint] *relationship*s within the loop is irrelevant; however, any implementation of the Perfect Joint Joint reduction, must be consistent in its [joint] *relationship* selection, as to enforce an equivalent operation across the PBSHM network.



(c) Step 3: Remove the previously selected [joint] *relationship* from the model.

Figure 5.7: The stages of a Perfect Joint Joint Canonical Form reduction against an Irreducible Element model graph. By performing this reduction, an unrequired loop is removed from the graph without the loss of any embedded knowledge within the model.

relationship loop to a single [perfect] and [joint] *relationship* as the physics of the interaction have been duplicated within the model; thus, one of the [joint] *relationship*s can safely be removed from the model without losing any structure knowledge regarding the interaction. The *Perfect Joint Joint* reduction rule also

simplifies the topology of the graph by removing another unnecessary loop.

Figure 5.7 illustrates the process of finding the [perfect], [joint], [joint] *relationship* loop, selecting one of the [joint] *relationship*s to remove, and finally removing the selected [joint] *relationship* from the graph. Whilst the *Perfect Joint Joint* reduction rule does not enforce which of the [joint] *relationship*s should be removed from the graph, any implementation of the *Perfect Joint Joint* reduction rule must be consistent in which [joint] *relationship* the algorithm decides to remove; if the same graph is reduced by a CFRR implementation, it must choose the same [joint] *relationship* to remove, each and every time i.e. in a planar graph, the 'right' joint is removed.

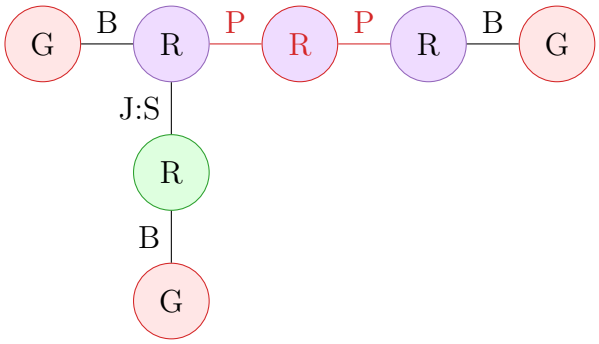
Perfect Relationships

The third rule proposed within the CFRR for now, is that any [regular] *element*, with *exactly* two [perfect] *relationship*s may be removed from the graph with the associated [perfect] *relationship*s, by creating a new [perfect] *relationship* between the neighbouring [regular] *elements* and migrating any knowledge within the [regular] *element* to be removed to the neighbouring [regular] *element*s.

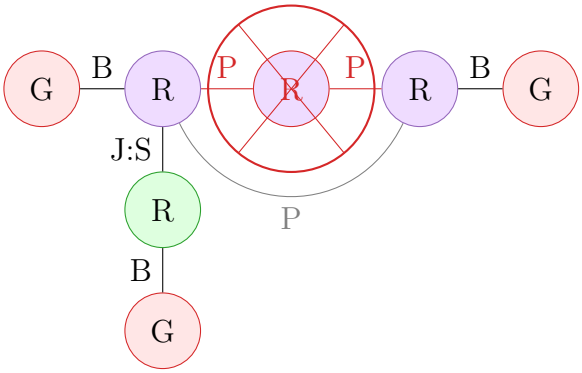
[Perfect] *relationship*s by their own definition, are present within an IE model where a larger component has been divided up into additional [regular] *element*s, either for representing a complex geometrical shape, or for the purpose of damage localisation within the model. In both of the aforementioned scenarios, the [perfect] [relationship] is only present within the model to handle model subjectivity or SHM necessity of the creator. Embedding complex geometrical shapes is important to gain advanced knowledge of the form of a component; however, such detailed knowledge is potentially irrelevant when trying to compare the overall similarity of two structures, but becomes increasingly relevant when trying to compare the similarity of structure subsections or validate the comparisons to a third party. The same premise holds true for division which has occurred because of damage localisation: knowledge on where damage has transpired within the model is vitally important for the author of the model, or when trying to relay knowledge back to the owner or operator; however, these details are irrelevant when determining the similarity of structures.

The *Perfect Relationship* reduction rule can safely reduce a [regular] *element* with two — and only two — [perfect] *relationship*s, as the knowledge contained within

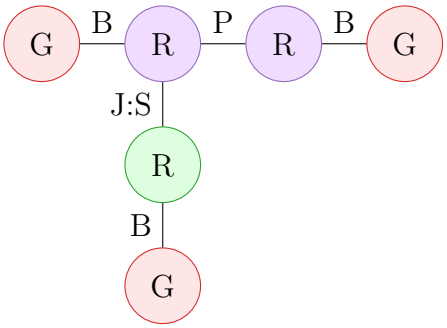
the selected [perfect] *relationship*s and associated [regular] *element* is irrelevant for similarity purposes, and can be merged into neighbouring [regular] *element*s without losing any structural relevant knowledge in the context of the *network*.



(a) Step 1: Select any [regular] *element* within the graph that has two — and only two — [perfect] *relationship*s. The pattern is highlighted in red.



(b) Step 2: Isolate the previously selected [regular] *element* and two [perfect] *relationship*s from the graph. Create a new [perfect] *relationship* between the two neighbouring [regular] *element*s from the isolated [regular] *element*.



(c) Step 3: Remove the previous-selected [regular] *element* and two [perfect] *relationship*s from the model.

Figure 5.8: The stages of a Perfect-Perfect Canonical Form reduction against an Irreducible Element model graph. By performing this reduction, an unrequired node is removed from the graph without the loss of any embedded knowledge required for similarity matching. Iterating over the graph with this reduction rule until no further *regular element*s are removed, will remove the unrequired sequences of repeated [regular] *element*s and [perfect] *relationship*s from the graph.

Figure 5.8 illustrates the process of finding the [regular] *element* with two — and only two — [perfect] *relationship*s, creating a new [perfect] *relationship* between the neighbouring [regular] *element*s of the selected [regular] *element*, and removing the original selected [regular] *element* and the associated redundant [perfect] *relationship*s from the graph. Whilst the *Perfect Relationship* reduction rule does not explicitly enforce that neighbouring [regular] *element*s must obey the [perfect] *relationship* matching type rule defined in Section 3.6, it is expected that any implementation of the CFRR, would ensure that the neighbouring [regular] *element*s of the selected [regular] *element* have matching values for the *contextual*, *geometrical* and *material* types before actioning the defined reduction rule.

It is envisioned that the [perfect] *relationship* reduction rule will be refined in the future to handle [regular] *element*s that have more than two [perfect] *relationship*s. In the fullness of time, the CFRR will have additional rules included to facilitate the removal of all unrequired variations within the *network*. In the final version of the CFRR, there will be no [perfect] *relationship*s present in a Canonical Form IE model; however, this statement will not be valid within the remit of the *Reality Model* (see Section 5.2.3).

5.2.2 Jaccard index results

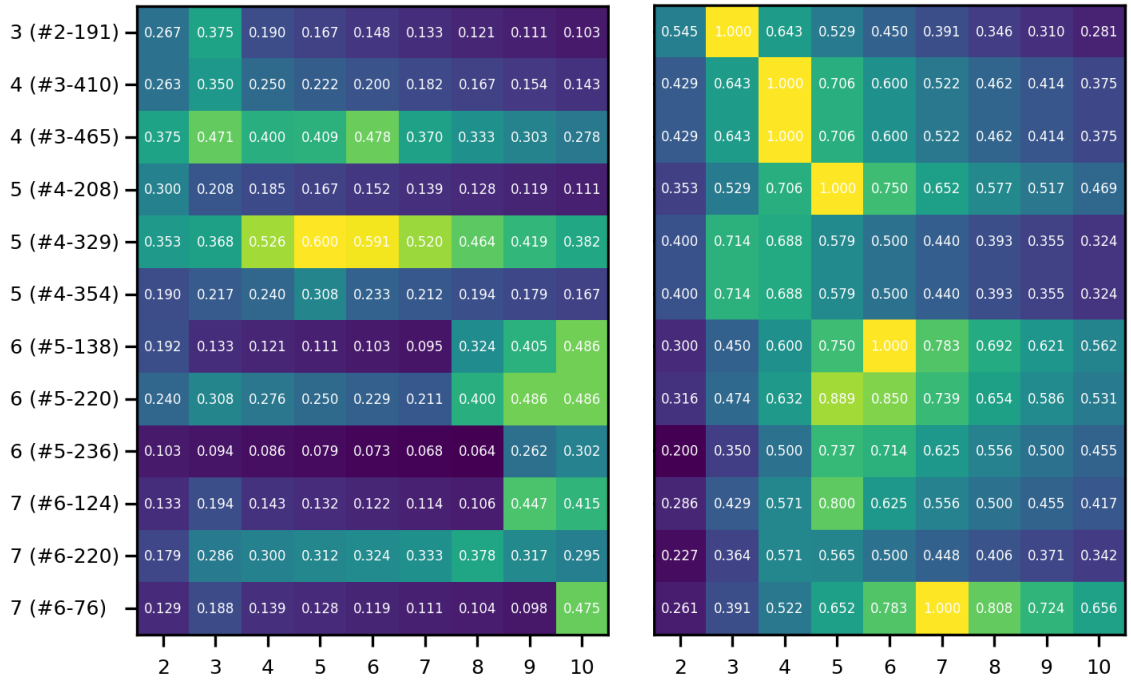
As discussed earlier in the chapter (see Section 5.1), the Jaccard Index — or Jaccard similarity coefficient — is a method for measuring the similarity between two datasets. In the case of determining the similarity of IE models, the algorithm was used by Gosliga *et al.* [6, 38], to generate a similarity score between two attributed graphs (see Figure 5.9). The logic behind the Jaccard Index boils down to calculating the intersection between G_1 and G_2, over the union of G_1 and G_2:

$$p(A,B) = \frac{|A| \cap |B|}{|A| \cup |B|} = \frac{|A| \cap |B|}{|A| + |B| - |A| \cap |B|} \tag{5.1}$$

The output from the Jaccard Index is a similarity score between 0 and 1, where 1 is similar and 0 is dissimilar. The calculation of the MCS between G_1 and G_2, is implemented via a backtracking algorithm — the Bron-Kerboshs algorithm to be precise — to find the largest common subgraph between two graphs — G_1 and G_2 in this case. In the interest of brevity, the logic of implementing the backtracking

algorithm is excluded from this chapter, the interested reader is recommended to read the original paper by Bron and Kerbosh [28] to understand the finer working of the algorithm.

Figure 5.9 displays the similarity matrix results using the Jaccard similarity coefficient against the matching dataset used in Figure 5.4 and the known *Canonical*



(a) The Jaccard Index similarity matrix results using the *detailed* Irreducible Element model without the *Canonical Form Reduction Rules*.

(b) The Jaccard Index similarity matrix results using the *Canonical Form Reduction Rules* to reduce the *detailed* Irreducible Element model before comparison.

Figure 5.9: The Jaccard Index similarity matrix when comparing the matching dataset to the known *Canonical Form* dataset using both the Jaccard Index without the *Canonical Form Reduction Rules* (see Figure 5.9a) and then with the *Canonical Form Reduction Rules* (see Figure 5.9b). The Attributed Graph contains only the embedding of the [regular] *element*s contextual type as a node attribute to keep results in direct comparison to Figure 5.4. The X axis are labelled with the number of spans of the Canonical Form graph, the Y axis are labelled with the number of spans the graph is associated with and the IE of the graph from the matching dataset. The label for the Y axis is missing from the second figure, because the labels are the same as in the first figure.

Form dataset for bridges with spans from 3-7. The ideal scenario for these similarity metrics, is that a bridge from the matching dataset should match as near identical — a value as close to 1 as possible — to the known *Canonical Form* bridge with the same number of spans. The similarity value should decrease in value the further away the number of spans being compared.

The first results in Figure 5.9a show the similarities when using none of the *Canonical Form Reduction Rules* (CFRR), and instead using the *Canonical Form* as a common form for comparison against. As the reader can evaluate, the Jaccard similarity coefficient is unable to find any discernible pattern between the matching dataset and the *Canonical Form* dataset. The second results in Figure 5.9b show the similarities when the matching dataset — containing *detailed* IE models — has first been reduced using an implementation of the CFRR before being evaluated against the common form *Canonical Form* dataset, using the Jaccard similarity coefficient. As the reader can see, the implementation of the CFRR within the *network*, improves the indicated values with the desired pattern of similarity (results within the same span should match identically with similarity values gradually decreasing through the change in number of spans) starting to emerge when comparing the *matching* dataset to the *Canonical Form* dataset.

5.2.3 Reality model

An Irreducible Element (IE) model is only concerned with structural composition. The environment in which the IE model is placed, the operational constraints of the structure, and the concerns of a structure owner, are but three examples of knowledge that, while being vitally important in the overall makeup of a structures' health, are out of the remit for an IE model. The aforementioned missing knowledge provides critical context to conditions a structure must endure; as such, they are required to be included within the global scope of PBSHM, whilst still remaining out of bounds to the structural comparisons portion of the PBSHM architecture.

A new model is required to capture the circumstances in which a structure resides, an encapsulation of the world in which the structure lives: the *Reality Model*. This model doesn't invalidate any of the proceeding research on capturing the structural composition of a model or any of the defined shared-data domains: *network*, *framework* and *database*. Instead, the model builds upon and encapsulates all of

these PBSHM-defined fundamentals into a hierarchical overarching model.

A *Reality Model* by itself, won't be an official specification or list of requirements akin to the specification and language of an IE model: instead, the model will be the summation of all available knowledge on a structure: structural composition, channel values, extracted features, sensor network, environmental and operational variables, and damage localisation concerns, to name but a few. Figure 5.10 depicts the potential hierarchical knowledge areas within the *Reality Model*.

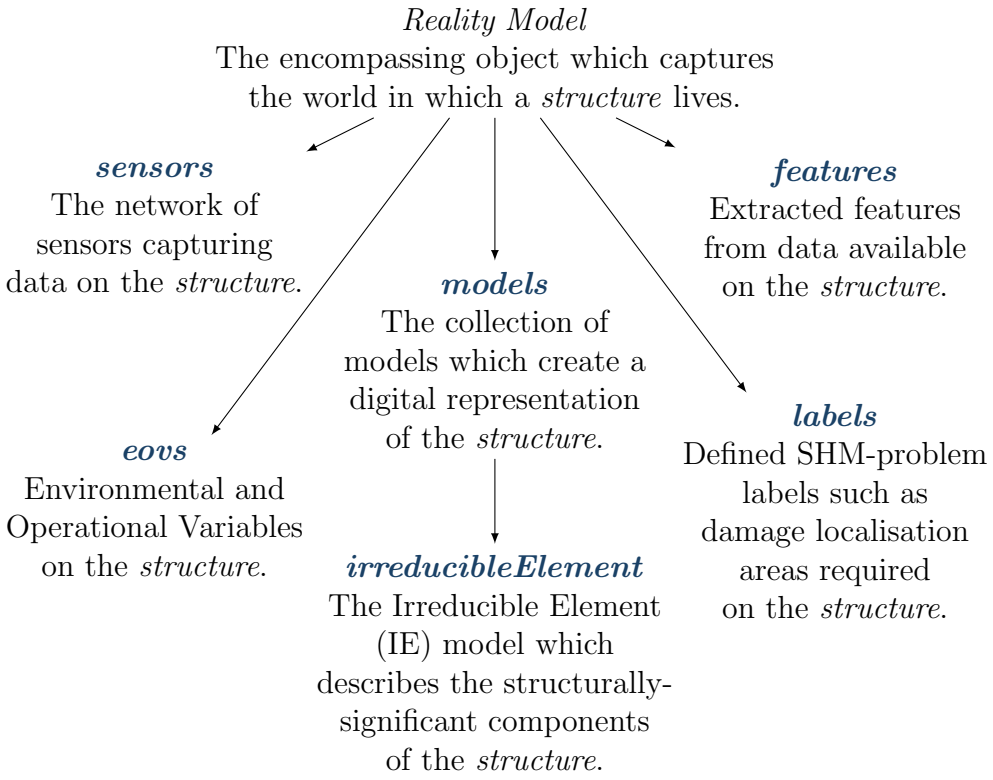


Figure 5.10: A selection of potential knowledge areas included within the hierarchical layout of the *Reality Model*.

The specifications and definitions of required knowledge, will be devolved to the individual areas of knowledge. The decisions as to what is required to capture structural composition, belong to an IE model, and as such are controlled by the IE model section within the PBSHM schema. The decisions as what is required to ensure a full picture of a sensor network, belong to the sensor network and as such will be defined by a future sensor network section within the PBSHM schema. It is only when the aforementioned knowledge areas are brought together, that the *Reality Model* achieves its full identity and has a powerful and meaningful purpose

within PBSHM.

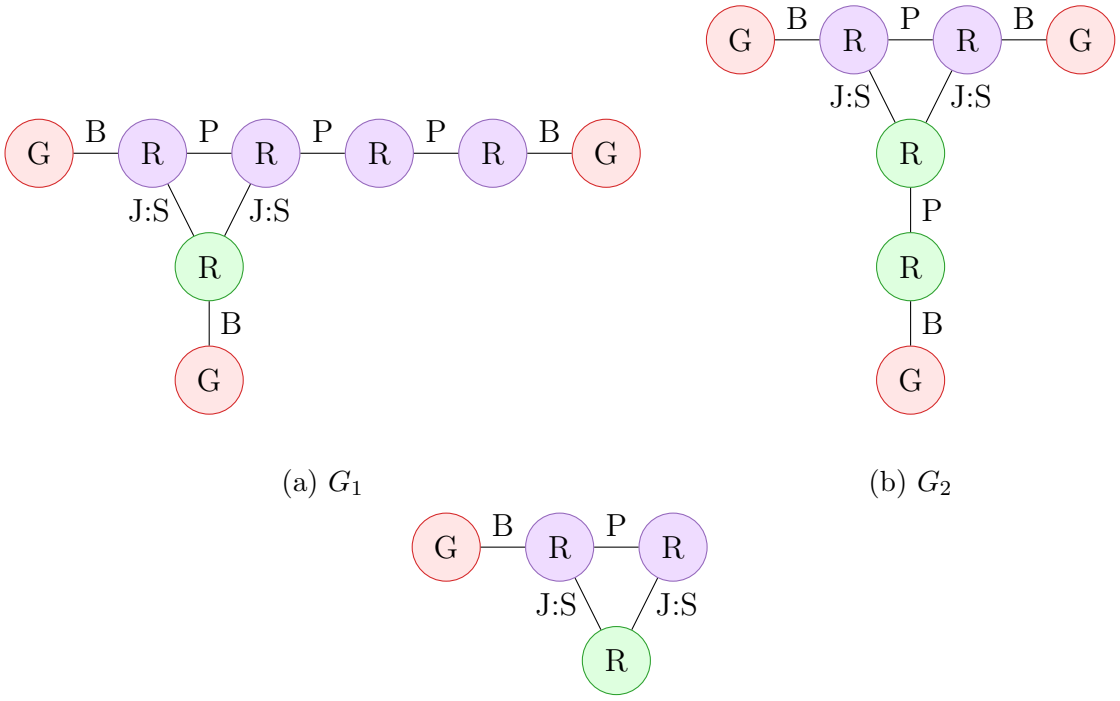
Whilst by the definition of a *Reality Model*, each knowledge area is devolved and has complete control of the associated data and language required to embed the associated required knowledge. The PBSHM shared data domain — *network*, *framework*, and *database* — must be aware of the *Reality Model* and understand how the presence of the model determines any confounding influences. The *database* will naturally become aware of any influences the *Reality Model* produces, by the expansion of new defined knowledge areas within the PBSHM schema. The *framework* will further organically expand to be *Reality Model*-aware, via the inclusion of new algorithms designed to process the enhanced available state of a structure, contained within the *database*.

Whilst the *network* operates its comparisons within the IE model domain, being *Reality Model*-aware means that additional restrictions may be required when considering the introduced *Canonical Form*. The whole purpose of an IE model and subsequently the *Canonical Form*, is to find similarities between structures, thus enabling new populations of structures to be established, and finally, learnt knowledge being transferred across the population. There is no point in attempting to transfer learnt knowledge across the population, if the knowledge being transferred is not applicable to the target structure because of the world in which the structure lives.

As such, each area of knowledge encompassed within the *Reality Model* must have the potential to restrict and inform the produced *Canonical Form* representation of a structure. This may be by the introduction of additional *Canonical Form Reduction Rules* which are only pertinent if certain data are present within the *Reality Model*, they may also be in the form of restrictions on when certain *Canonical Form Reduction Rules* can be applied. A specific value within the *labels* section of the *Reality Model* may dictate that certain *element*s are protected, thus they may not be removed from the model through the *Canonical Form Reduction Rules*. In a *network*, where only IE model data resides, the *Canonical Form* representation of two homogeneous structures, should be identical; however, when additional *Reality Model* data is included within the *network*, the *Canonical Form* representations of two homogeneous structures, may no longer be identical.

5.3 Graph Matching Network

The Jaccard Index is simply one methodology for generating a similarity between two sets of data, once one has established the known intersection between these two sets of data. The way in which this intersection has been found previously — within the context of a graph — is using the Maximum Common Subgraph (MCS). The MCS is an approach from Graph Theory [11, 12], where the goal is to find the largest shared graph between two graphs (see Figure 5.11). The problem with this approach, is that each node within G_1 and G_2, have to match exactly.



(a) G_1

(b) G_2

(c) MCS of G_1 and G_2 when considering both topology and the *contextual* type attribute.

Figure 5.11: The Maximum Common Subgraph (MCS) between G_1 and G_2, where the graphs are two bridge IE models with the *contextual* type from the [regular] *element* is embedded as an attribute within the associated nodes.

If one takes the example of *material* within a [regular] *element*, say a beam on a bridge. Both bridges are classified as two-span beam-and-slab bridges; however, in the first bridge, the beam has a *material* type set of 'metal' →'ferrousAlloy' →'steel', in the second bridge, the beam has a *material* type set to 'metal' →'aluminiumAlloy'. No matter how this *material* knowledge is encoded into an attributed graph, the

nodes of the corresponding [regular] *element*s would never be included within the MCS, without a decision to omit knowledge from the AG. To facilitate the inclusion of these nodes within the MCS, a decision would have to be made to only include the first level of *material* type within each node. Such modifications to knowledge encoding within the AG, necessitate futile knowledge of both the context in which the structures are based and the mechanics of the similarity metrics. Alternately, a method in which all available knowledge from the IE model can be encoded within the attributed graph, and then the similarity algorithm itself can determine which of these attributes are necessary for determining the similarity of the *network*.

Neural Networks [90] are a subset of machine learning algorithms aiming to replicate how the neurons inside the brain processed and passed data between themselves. If one examines the process of a Multi-layer Perceptron (MLP) [91] to establish the premise, the neural network — in the MLP case — works under the concept of layers of knowledge. Within each layer of the network is a number of neurons — in network terms, think of these as nodes — each neuron then connects to every other neuron in the next layer of the network — again in networks terms, think of them as links; however, there are no connections between neurons within the same layer. The first layer is called the *Input Layer*, and the sole purpose of this layer is to encode knowledge from a data point within the source dataset — in the case here, this would be the IE model knowledge — into its neurons as a numerical value. The last layer is called the *Output Layer*, this layer again has neurons like the *Input Layer*; however, the number of neurons within this layer will depend on the number of outputs or classifications desired for the neural network to make. In between the *Input Layer* and the *Output Layer* are the *Hidden Layers*, how many of these layers there are, and how many neurons inside each layer is part of the individual implementation of the neural network.

The aforementioned connections between neurons are where the machine learning part of the algorithm takes place within a neural network. Each connection is made up of *weights* and *biases*. These *weights* and *biases* enable multiplication and addition — respectively — of the value within a neuron. Each data point within the source dataset has a label defining what the correct output or classification should be against the neurons in the *Output Layer*. The source dataset is then divided into *training*, *test*, and *validation* subsets. Initially, the neural network is initialised with random *weights* and *biases*. One by one, each data point within the *training* portion of the source dataset, is fed into the neural network and the values of the *Output*

Layer are stored. A *Loss Function* is defined which decides if the neural network has performed well from the desired *Output Layer* values verses the actual *Output Layer* values. The *weights* and *biases* are adjusted with the goal of improving the results of the *Output Layer*, minimising the *Loss Function*.

Graph Neural Networks (GNN) [92] are a category of neural networks with the focus of naturally embedding graph data within the *Input Layer*. Li *et al.* [93] have recently introduced the Graph Matching Networks (GMN) derivative within the GNN domain, where instead of categorising data, the objective is to determine the similarity between graphs. The GMN can be trained in two ways; pairs of labelled graphs, or triplets of unlabelled graphs. In the first method, each graph within the dataset G_1, is paired with another graph within the dataset, G_2. If the graphs, G_1 and G_2 are determined to be similar, then a label of 1 is assigned to the pair; however, if the graphs are determined to be dissimilar, then a label of -1 is assigned to the pair.

$$(G_1, G_2) = t \, \epsilon \, \{-1, 1\} \tag{5.2}$$

In the second method of training the GMN, each graph, G_1, is paired with one graph within the dataset that it is similar, G_2, and one graph within the dataset that is dissimilar, G_3. The formed triplet doesn't require a label; however, it does require the order of the graphs within the triplet to be observed:

$$(G_1, G_2, G_3);\ G_1 \text{ is similar to } G_2,\ \text{ but } G_1 \text{ is dissimilar to } G_3 \tag{5.3}$$

The work outlined with this chapter has shown the potential of a common form within the PBSHM *network*. The main disadvantage with a method such as this, is manually learning and forming the *Canonical Form Reduction Rules* to reduce the *detailed* IE model down to the *Canonical Form* representation. The hope of using a method such as the GMN, is that the neural network in the code of the GMN, can learn yet unknown reductions. To evaluate the use case of a GMN within the context of the PBSHM *network*, one first must establish if the GMN can learn the similarity without using the common form.

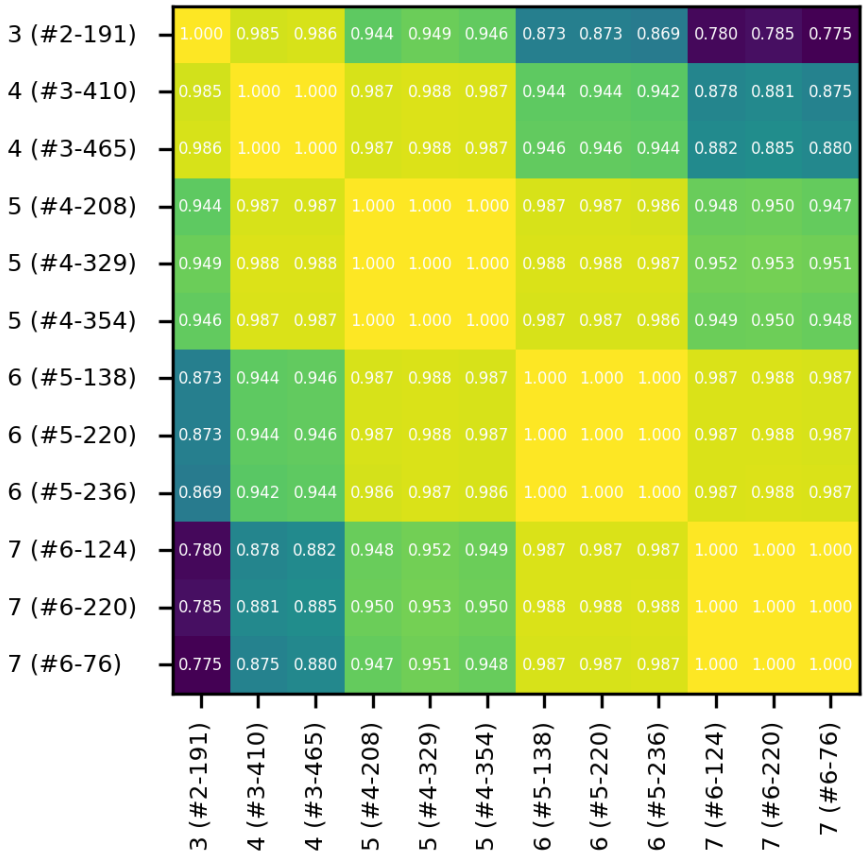
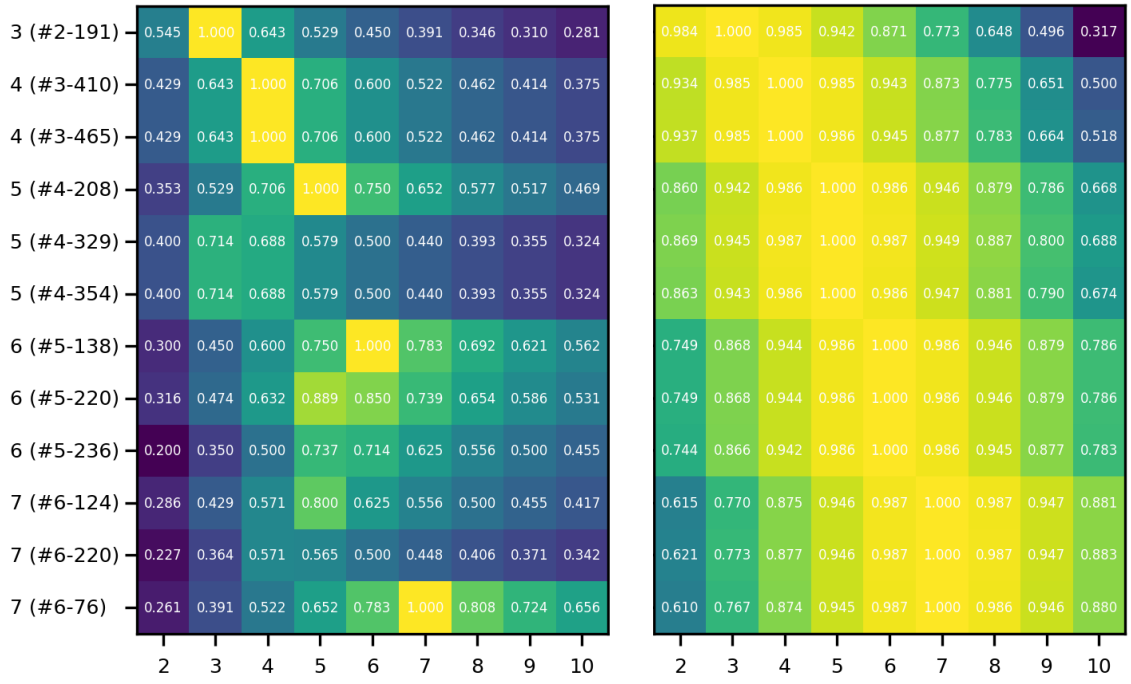


Figure 5.12: The Graph Matching Network similarity matrix results when comparing the *detailed* Irreducible Element model against itself. The axes are labelled with the number of spans the graph is associated with and the ID of the graph from within the dataset.

Figure 5.12 depicts the results of using the GMN against only the matching dataset. The GMN is trained using sets of labelled pairs, using a loss function of the margin-based Euclidean distance and utilising the Adam optimiser [94] for the minimisation of the loss function. As one can see, the GMN is able to learn and identify the beam-and-slab bridges of the same span as identical, with a result of 1. The GMN is also able to identify the desired tiered similarity, when travelling away from the number of spans. If one looks at the results for the six-span bridges, the bridges with the closest similarity are the group of six-span bridges. The bridges with the next-nearest similarity are the bridges with five and seven spans, then the four-span bridges and finally the three-span bridges. Whilst the results are not as separated in distance as the Jaccard Index results in Figure 5.9b, there is a small noticeable change in the results the further one moves away from the target span.



(a) The Jaccard Index similarity matrix results using the *Canonical Form Reduction Rules* to reduce the *detailed* Irreducible Element model before comparison. These are the same results as in Figure 5.9b.

(b) The Graph Matching Network similarity matrix results when comparing the *detailed* Irreducible Element model with the known *Canonical Form* representation.

Figure 5.13: The similarity matrix results for both the Jaccard Index (see Figure 5.13a) and the Graph Matching Network (see Figure 5.13b) when comparing the matching dataset — containing *detailed* Irreducible Element models — against the known *Canonical Form* dataset. For the Jaccard Index results, the *Canonical Form Reduction Rules* were used to reduce the *detailed* IE models before comparison. For the Graph Matching Network results, the Graph Matching Network learnt the reductions required against the training dataset — a labelled graph pairing of *detailed* Irreducible Element models and known *Canonical Form* representations. The Attributed Graphs for both algorithms, contain only the embedding of the [regular] *element*s contextual type as a node attribute to keep results in direct comparison to Figure 5.4 and 5.12. The X axes are labelled with the number of spans of the Canonical Form graph, the Y axes are labelled with the number of spans the graph is associated with and the IE of the graph from the matching dataset. The label for the Y axis is missing from the second figure, because the labels are the same as the first figure.

Figure 5.13b shows the results of introducing the *Canonical Form* representation into the GMN comparisons; instead of the GMN learning the reductions needed between *detailed* IE models, the GMN learns the reductions required to reduce the *detailed* IE model to the *Canonical Form* representations. As one can see, the GMN is still able to identify *detailed* IE models to the *Canonical Form* representation with the same number of spans as identical. The results also show that the pattern of similarity decreasing with neighbouring number of spans from the target span, is also preserved. These results illustrate the flexibility of the GMN, the algorithm is able to learn the reduction rules between *detailed* IE model to *detailed* IE model, or from *detailed* IE model to the *Canonical Form* representation.

Figure 5.13 illustrates the results of comparing the performance of the Jaccard Index using the CFRR (see Figure 5.13a), verses the GMN comparing the matching dataset to the known *Canonical Form* representation dataset (see Figure 5.13b). From the initial inspection of the results, it is clear to see that the GMN algorithm outperforms the Jaccard Index with CFRR when considering the ability to identify a pattern of similarity within the example *network*; however, when one considers the context of the algorithms, the outcome is not so clear.

If one looks at the comparisons for the bridge *7 (#6-124)*, the Jaccard Index with CFRR incorrectly identifies the five-span *Canonical Form* representation as the closet match to the input bridge, whereas with the GMN, the algorithm correctly identifies the seven-span *Canonical Form* representation as the closest match. The GMN is evidently — within the context of the example scenario — able to learn reduction rules which are not currently understood or implemented within the CFRR; this may lead one to imagine, that the GMN algorithm should be used above the Jaccard Index with CFRR; however, to achieve this learnt knowledge, a not insignificant amount of bridges were required for the GMN to build the aforementioned knowledge. In direct comparison, the Jaccard Index with CFRR required no previous examples of similar bridges before it could establish a similarity.

Without modification to the existing GMN algorithm, there is no methodology of extracting which *element*s or *relationship*s cause the similarity, thus providing a stumbling block in the algorithms' ability to communicate back to a *framework* user, why the given similarity is thus. The Jaccard Index with CFRR; however, is able to communicate back to a *framework* user, as to where the similarity has been established, via the MCS. Both of the aforementioned algorithms are able to generate a similarity within the *network*: and as such belong within the *framework*.

When each algorithm should be used, perhaps, requires a larger viewpoint of the lifecycle of PBSHM.

Whilst PBSHM is still within its infancy, one cannot rely upon the *network* having existing examples to generate learnt knowledge; instead the *network* will need to depend upon algorithms which require no previous examples to learn from, such as the Jaccard Index with CFRR. Once PBSHM has established itself to the extent of having multiple examples of a single type of structure, learning algorithms such as the GMN will have their place within PBSHM. The problem of data availability should not block research into learning algorithms, on the contrary, research should continue into machine learning approaches — using simulated datasets — and focus on identifying what knowledge can be extracted from these approaches, and incorporated back into the global knowledge of similarity and processes such as the *Canonical Form Reduction Rules*.

5.4 Conclusion

This chapter has shown that the variations present within the *network* because of author bias, directly effect the computed similarity scores when using a graph-theory-based calculation. The chapter introduced the *Canonical Form* as the vehicle within PBSHM to remove these variations from the *network*. A *detailed* author-generated IE model is submitted into the *network*, the *Canonical Form Reduction Rules* (CFRR) reduce the detailed IE model into the Canonical Form representation so that no variations are present within the model, but all structural knowledge which is relevant to the similarity comparisons is retained.

The first three CFRR are introduced; however, these rules are intended to be expanded upon in the future as further knowledge is obtained upon what knowledge is crucial for the *network*. The Maximum Common Subgraph (MCS) and Jaccard Index (JI) is utilised as a graph-theory-based similarity metric to evaluate the use of the CFRR. While using no CFRR, the algorithm is unable to detect any noticeable pattern of similarity within the input graphs; however, when the CFRR are used to reduce the input graphs before comparison, a pattern of similarity starts to appear.

The *Reality Model* is introduced as the vehicle within PBSHM to encapsulate the knowledge regarding the world in which the IE model is placed. While this model is

important within the wider context of PBSHM, there is a direct consequence that the Reality Model has upon this chapter. As stated, PBSHM does not invalidate the previous rules and desires of SHM; as such, SHM-driven labels may end up determining what *element*s may or may not be reduced from a model, thus will directly impact upon the CFRR.

Lastly, the chapter evaluates the use of a machine-learning approach to deriving the similarity within a population. The Graph Matching Network (GMN) algorithm is used in comparison to the MCS and JI method described above. The GMN is able to find the similarity patterns and identify potential reductions, which were not previously know when using the CFRR approach.

Chapter 6

From Bridges to Aeroplanes

Throughout this thesis, bridges have often been used to illustrate the introduced transformations within the language and context of Irreducible Element (IE) models. Whilst civil structures — such as bridges — are one of the desired structure categories to be included within the PBSHM database, they are by no means the only such category which are desired to reside within the PBSHM database. This chapter looks at using the new IE model language in the context of aeroplanes.

Bridges and aeroplanes can often apply commonality of properties across their IE models, such as utilising the same metal within their construction; however, each structure has a unique purpose and as such, has unique characteristics which require capturing within an IE model — aside from the fact that one of them defies gravity! With each purpose of a structure can often come unique construction techniques. Rules defined for the creation of bridge IE models, cannot be easily transferred to the curation of aeroplane IE models; consequently, rules are required for the process of breaking down an aeroplane's components into the corresponding *element*s and *relationship*s for IE model production.

This chapter firstly explores the differences in construction techniques between bridges and aeroplanes and the associated consequences for IE model production in Section 6.1, introduces a new graph notation for PBSHM structures via the use of a simplified laboratory structure in Section 6.2 and finally applies these learnt changes to a real world ex-Royal Air Force Hawk T.Mk1 in Section 6.3.

6.1 Where is an Aeroplane not a Bridge?

Bridges – in the most part – are constructed to bear a weight along a defined path over an obstacle. Aeroplanes are constructed to carry a load from one point to another by breaking the earth's gravitation pull. During the lifecycle of an aeroplane, the geographical positioning of the structure will change during its operation, while maintaining a healthy classification state; whereas, a bridge would be often classified as unhealthy, if the geographical positioning varied during operation.

These differences in designated purpose between aeroplanes and bridges, often lead to contrasting methodologies for their construction; subsequently, these methodologies and design choices must be respected and captured during the formation of any associated IE models. This is not to say that there would not be any overlap between bridges and aeroplanes; on the contrary, there may be much commonality of properties across both categories of structures. If one considers the materials used: steel and aluminium are both materials that would be seen as normal if present within either category of structure; however, how each category uses these materials is subtly different. There are other properties which would be classed as unusual if present within either structure: an aeroplane containing concrete or a bridge designed with aerofoils are just two examples.

These examples highlight why the introduced restructuring of the IE model language is fundamental for embedding different types of structures within PBSHM. The purpose, form, material and positional knowledge on an *element* are now partitioned into their designated areas, this separation of duties facilitates the shared properties to be present; whilst permitting important structural differences to not be overlooked or hidden because of a conflicting property.

The process of generating an IE model of a structure can be broken down into two separate processes; determining how to divide a structure into structurally-significant components, and then embedding the available knowledge regarding each of these component and their interaction into the model. The restructured language of IE models only effects the second part of this process — the knowledge embedding. Decisions and reasoning behind what divisions have occurred are captured from the first part of the process, but the determination of how to divide a structure, remain valid from the rules introduced by Gosliga *et al.* [6]. Whilst these division rules remain valid, they were initially formulated against a use case of bridges [38];

therefore, this process required reevaluation against the context of aeroplanes.

One of the defining differences between the construction of an aeroplane and a bridge, is that with bridges, the structure can often be broken down into common known components which themselves give defined meaning towards the complex form taken by whole physical component. If one takes for example an I-beam: the name alone gives meaning to the shape and construction of the component. Additionally, the shape is standardised and has a set of predefined accepted dimensions required to capture any variations. Whereas, if one considers the fuselage of an aeroplane, there are no defined shapes which can universally describe its complex form and no known set of dimensions which would encapsulate knowledge for all possible variations. As such, a different method for deconstructing the object — an aeroplane — is required in this instance.

In general, fuselages are made by wrapping a protective layer of material around a hollow interior; thus creating a shell. One could choose to represent the fuselage via a series of [regular] *element*s with a *geometrical* type of 'plate' →'other' declarations — given appropriate *relationship*s between *element*s — with custom dimensions to encapsulate the bending. This approach may provide an accurate

Figure 6.1: Side view of the Hawk T.Mk1 fuselage at the Laboratory for Verification and Validation. A section of the fuselage is enlarged to depict the joints between two metal sheets.

description of how the component was manufactured; however, it does not provide a meaningful knowledge of the form of the component. In a crude IE model, one could potentially represent this whole component by declaring a [regular] *element* with a *geometrical* type of 'shell' →'translate' →'cylinder'; however, the model would not be truly representative of the form of the fuselage, let alone supporting any form of information regarding damage localisation within the structure. Instead, a methodology is proposed that both captures the form of the component, and also embeds the knowledge of component manufacturing into the model.

If instead of breaking the structure down into components, one instead breaks the structure down to the main sub-structures of an aeroplane: fuselage, wings, horizontal stabilisers, vertical stabiliser and landing gear. One can then start to break the sub-structures down into separate components using any visible construction markers or structure features as a guide for where division should take place. The goal is to try and 'cut' the subsections up into smaller components, which can then be modelled as [regular] *element*s within the model. Markers for division may be as simple as a weld or rivet line where two metal sheets have been joined together, they could be where a large geometrical change is present, or they could simply be where a gap is present to enable movement within the structure. These markers create natural 'lines' on the structure for facilitating [regular] *element* boundaries throughout.

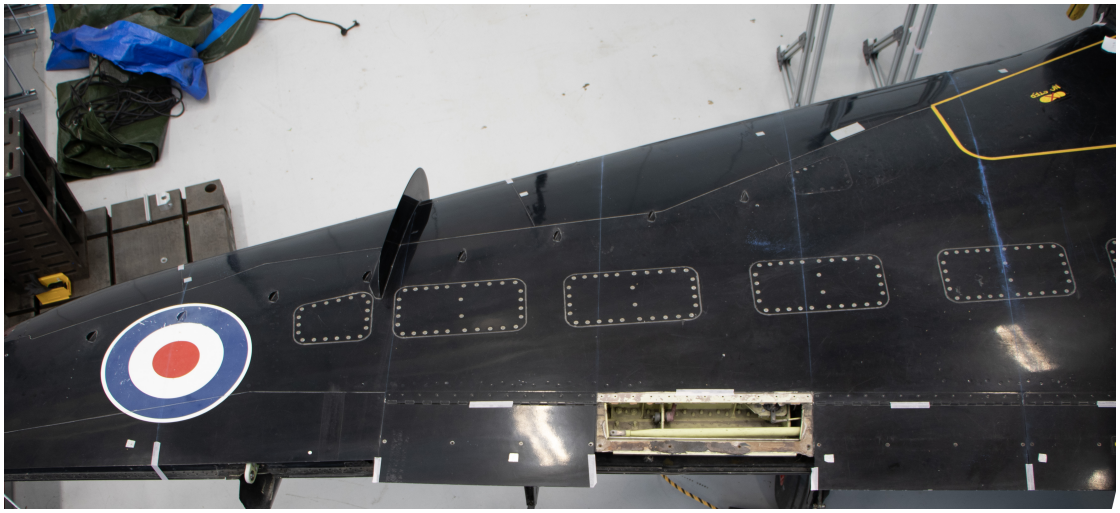


Figure 6.2: Left-wing view of the Hawk T.Mk1 fuselage at the Laboratory for Verification and Validation. The blue chalk lines on the wing indicate the subdivision obtained by using the features of the flaps as a natural line.

If one takes the example of the fuselage of the Hawk T.Mk1 pictured in Figure 6.1. One can see from the enlarged section on the fuselage, that there are natural lines present because of weld and rivet lines present from the construction of the aircraft. There are also large geometrical changes where the cockpit is present along the fuselage. Using these natural lines as boundaries between [regular] *element*s would honour the manufacturing techniques while capturing the complex geometrical form within the model.

This process can be further expanded when considering the wing of the Hawk pictured in Figure 6.2. Not only are there natural lines present because of the placement of hinges to facilitate flap movement, but the essence of the movement being present within the wing, provides a natural division from observing this structural feature of the 'flaps'. The blue chalk line in Figure 6.2 shows the division of the wing across the X axis when using the 'flaps' as a structural feature for dividing up the wing. Further details on the specific division of subcomponents in the Hawk is given in Section 6.3.

6.2 The GARTEUR Structure

The GARTEUR SM-AG19 structure is a laboratory test bed designed by the Structures and Material Action Group (SM-AG19) of the Group for Aeronautical Research and Technology in EURope (GARTEUR). The aim of the group was to evaluate the performance of different ground vibration tests [95–97] on aeroplanes. To facilitate a standardised test setup, a common test bed was build to mimic the dynamic response of an aircraft: the GARTEUR SM-AG19 test-bed – henceforth known as the GARTEUR structure.

Figure 6.3 shows a picture of a GARTEUR type structure in the Laboratory for Verification and Validation [98]. The construction of the structure is simple in nature; it consists of a fuselage, wings, winglets, and vertical and horizontal stabilisers; however, unlike most real-world aircraft, each sub-structure present within the aircraft consists of only a single uniform beam or plate. One could state that the construction techniques used within the GARTEUR structure are more akin to the construction techniques found within a bridge, than an aeroplane.

As discussed earlier in the thesis, an IE model may not be the final form in which

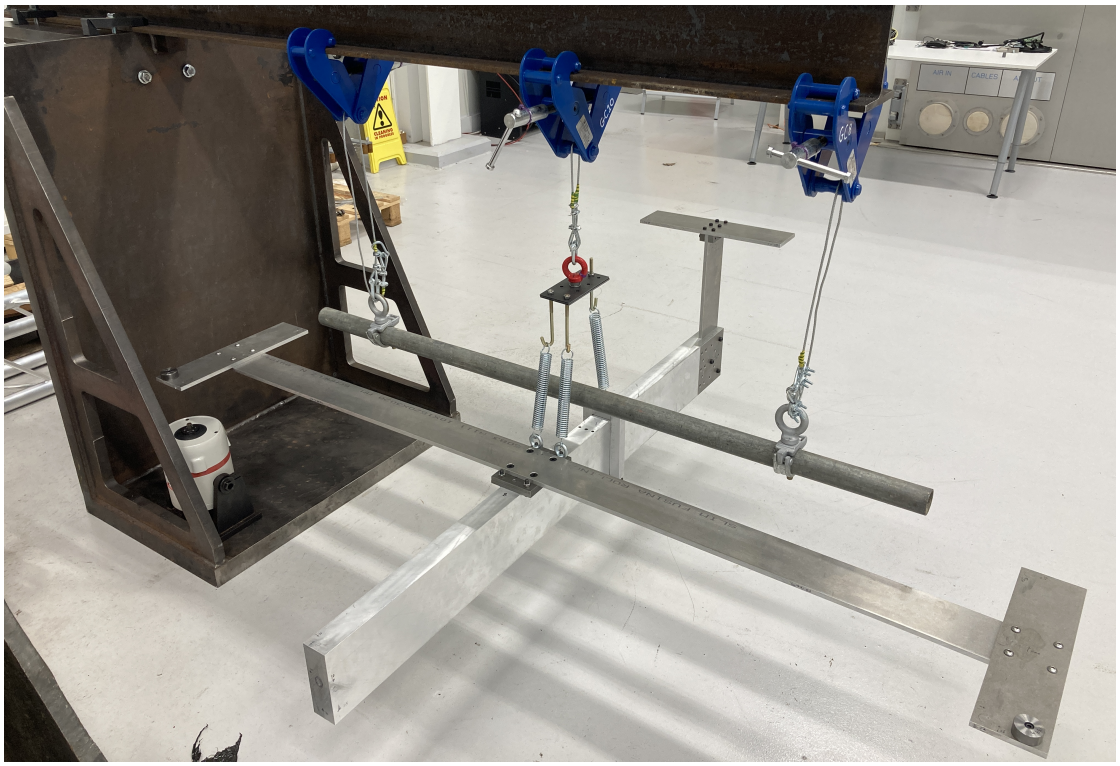


Figure 6.3: A photo of a GARTEUR SM-AG19 test-bed structure in the Laboratory for Verification and Validation.

PBSHM data resides: the PBSHM network may necessitate that an IE model be converted into the associated AG representation. While depicting the AG representation of a simple IE model such as a simplified two-span beam-and-slab bridge is rudimentary, the same cannot be stated when considering such complex models as seen from real-world structures. A simplified notation is required for relaying the detailed information present within an IE model in an AG form. The GARTEUR structure is used within this chapter as a bridge structure — pardon the pun — from simplified theoretical structures to real-world structures in the context of introducing the proposed PBSHM AG notation.

Using the newly introduced [perfect] *relationship* rules, the GARTEUR structure could be broken down into any number of [regular] *element*s; however, for the purpose of introducing the AG notation, the following division of sub-structures has occurred:

- *fuselage*: three [regular] *element*s before the interaction between the *fuselage* and the *wing* section, one [regular] *element* for the interaction between the

fuselage and *wing*, and three [regular] *element*s after the *wing* interaction but before the *tail* interaction, and finally one [regular] *element* where the interaction between the *fuselage* and *vertical stabiliser* is present.

- *wing*: three [regular] *element*s on either side of the *wing*s between the *fuselage* interaction and the *winglets*, one [regular] *element* for the interaction between the *fuselage* and the *wing*.
- *winglet*: three [regular] *element*s for each *winglet* with the centre *element* interacting with the *wing*.
- *vertical stabiliser*: three [regular] *element*s, with the bottom *element* interacting with the *fuselage* and the top *element* interacting with the *horizontal stabiliser*.
- *horizontal stabiliser*: three [regular] *element*s, with the center *element* interacting with the *horizontal stabiliser*.

The structure is considered as a [free] *model* with [perfect] *relationship*s between the *regular element*s within a sub-structure, and [joint] *relationship*s with a [static] *nature* to be present when sub-structures are interacting with each other.

6.2.1 Attributed graph notation

When one starts to study graphs from real-world structures, there are often repeated vertices or edges present within the graph. This scenario is by no means alarming and would be considered ordinary when one considers the reasoning behind the repetition. The inclusion of [perfect] *element*s within the language of IE models implies that the user has divided up a larger *element* into smaller components for any of the reasons set out in this thesis: damage localisation and geometrical knowledge embedded are but two of the potential reasons. This scenario will often necessitate that vertices be present within the associated graph which are repeated with the same attributes and respective edges.

Another scenario is that a sub-structure will be repeated numerous times within the structure. One example of this is the landing gear on an aircraft. One could state that the landing gear on the left wing of the aircraft is the same set of components as used on the right wing. This causes the same pattern — or subgraph — to appear

multiple times within an IE model graph. One could further expand this scenario and say that certain sections of a structure are just mirrored copies from another place within the structure, say the wings on an aeroplane: it could be stated that the left wing is simply a mirror of the right wing.

While these repetitions within a graph are expected and indeed advantageous within the context of similarity generation within the *network*; they prove troublesome when attempting to communicate a model's graph representation in pictorial form for a real-world structure, because of the space required often being larger than the available medium. The new notation rules within this section of the chapter, aim to aid representation of real-world structures within the pictorial form required for documentation.

The first repetition one sees within IE model AGs, is when the same pattern of vertex and edges are repeated multiple times within a graph. Figure 6.4 shows four [regular] *element*s and their associated edges. Two of these *element*s have identical attributes and matching number of edges — with identical attributes. These vertices — highlighted in red — and edge combinations can be replaced by a single vertex with a dashed line around the vertex and dashed edges instead of solid lines. The number of vertices represented by this single vertex, is displayed as an integer value.

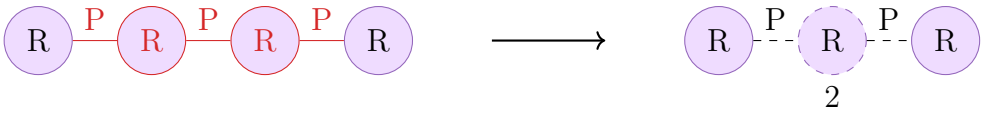


Figure 6.4: In the left graph, there are two vertices which are repeated with the same attributes and edges — highlighted in red. These vertices can be replaced with a single vertex which instead of having a solid circle around the vertex, has a dashed line around the vertex and the edges are replaced with dashed lines also. The number of vertices it represents is displayed either below or to the side of the vertex.

The second repetition one sees within IE model AGs, is when a pattern of related vertices and edges are repeated multiple times within a graph. Figure 6.5 shows a graph with a repeated subgraph on either side of the centre vertex — highlighted in red. Each of the subgraphs on either side of the centre vertex have identical vertex and edge attributes. This repetition within the graph can be replaced by selecting one of the instances of the subgraph and providing a reference for the subgraph — S_{lw} in the example displayed in Figure 6.5 — and denoting the subgraph reference by drawing a dashed box around the associated vertices and edges. Any other

presences of this subgraph within the graph can be replaced with a square dashed vertex with the reference of the subgraph it refers to: S_{lw}. If an instance of the referenced subgraph is determined to be a symmetrical instance of the referenced subgraph, then the superscript M is to be used to denote that this instance is a mirror of the referenced graph: S_{lw}^{M}.

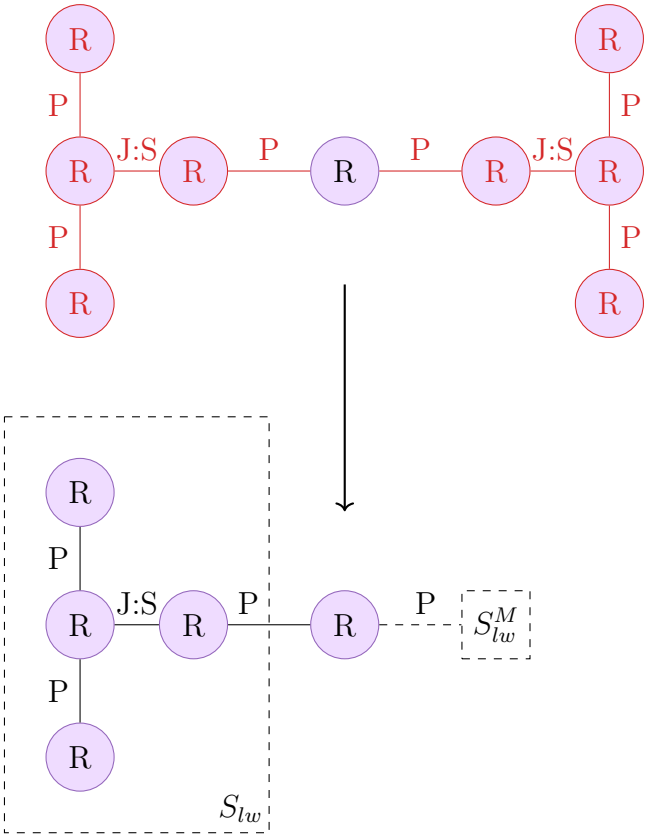


Figure 6.5: In the top graph, there is a repetition of a subgraph on both the left and right sides of the centre vertex — highlighted in red. This can be simplified by drawing a dashed line around an instance of the subgraph and providing a label for that subgraph. Subsequently, any further instances of this subgraph can use the provided reference in a square dashed vertex to denote the presence of a referenced subgraph.

The aforementioned notation can therefore be used to provide a succinct graph notation for real-world structures. An AG representation for the described IE model of the GARTEUR structure is provided in Figure 6.6 using the proposed notation. In the interests of brevity, the AG representation of the IE model without using the described notation is not included within the body of this thesis; however, the interested reader can find this within the appendix as Figure B.1.

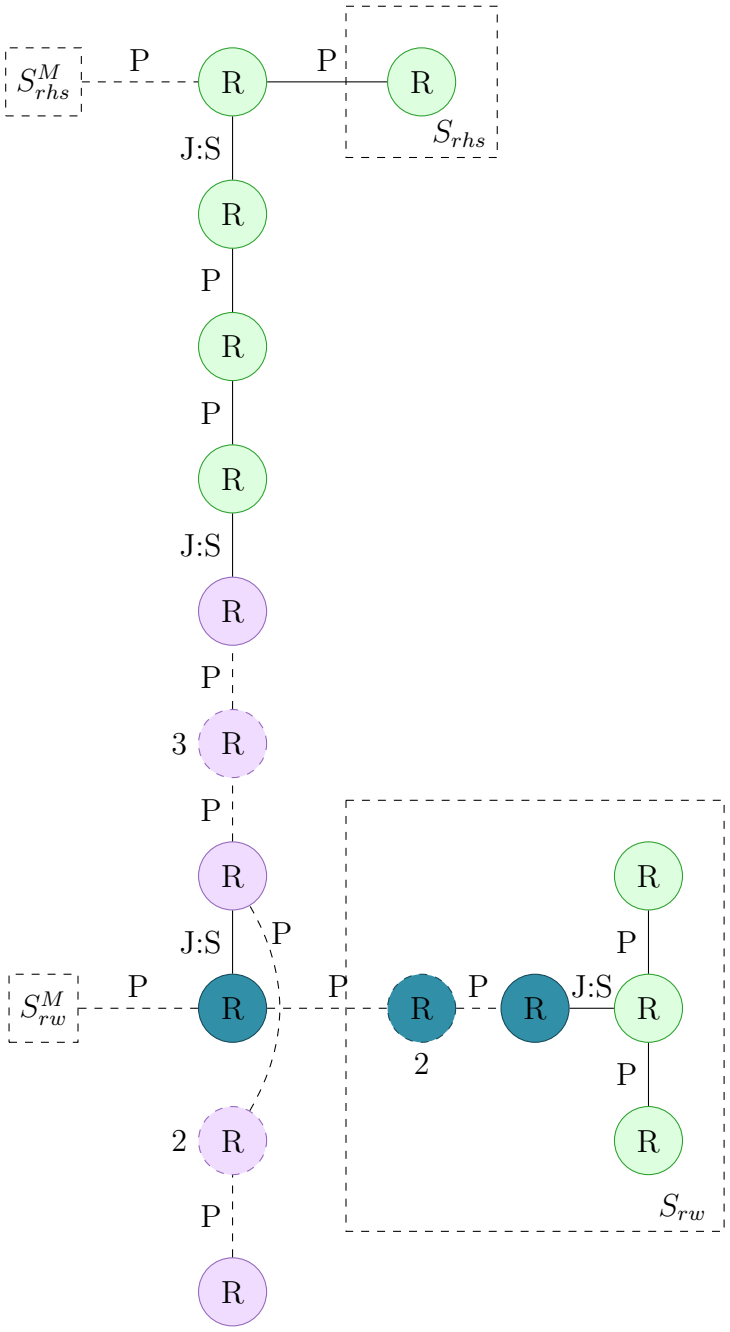


Figure 6.6: The AG representation of the GARTEUR IE model described within this chapter using the proposed PBSHM AG notations.

6.3 Hawk T.Mk1

The Hawk T.Mk1. [99, 100] is a fast-jet aeroplane used by the Royal Air Force (RAF) [101] for training pilots. Over the years, the aeroplane has served dual roles for the RAF: originally as a training aircraft for new pilots before graduating onto other fast-jets, and in an aerobatic role under the 'Red Arrows'. Whilst the Hawk T.Mk1 is being retired from its active trainer role by the RAF, the aeroplane still serves as a good example — within the context of PBSHM — of how to deconstruct a relatively modern aeroplane for the purpose of IE model generation.

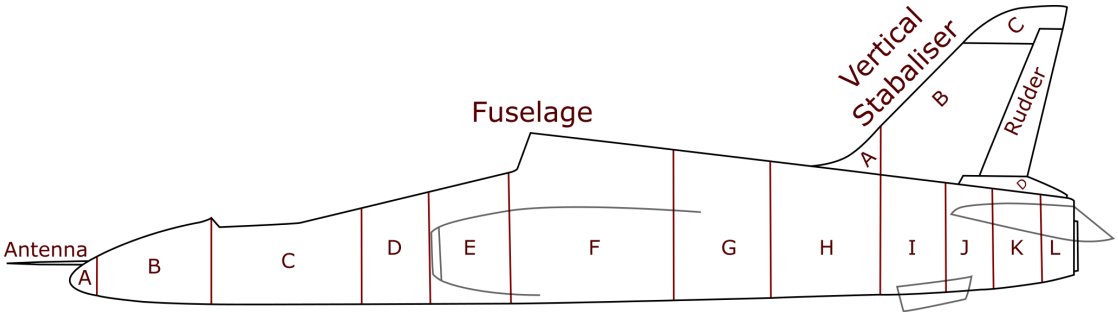
This section of the chapter looks at applying the methodology discussed in Section 6.1 to a real world Hawk. The specific Hawk in question is *XX184* and currently resides at the Laboratory for Verification and Validation [98] in Sheffield, UK. Whilst this instance of the Hawk pictured in Figure 6.1 and Figure 6.2 has been used to generate the IE model included within this chapter, it should be noted that the IE model generated is for the generalised form of a Hawk T.Mk1, as *XX184* has several components missing from the airframe (as pictured in Figure 6.2) because of experiments and testing performed on the airframe before the IE model was generated. Where components are missing, components have been inferred from a mirror subcomponent — component missing on the left wing can be inferred from components present on the right wing — or from the technical drawings by Rolfe [100].

For the purpose of IE model generation, the Hawk can be broken down into nine main sub-structures: fuselage, vertical stabiliser, left wing, right wing, left horizontal stabiliser, right horizontal stabiliser, left landing gear, right landing gear, and nose landing gear. Because of the inherent manufacturing variations, each Hawk will be unique in its own exact dimensions; however, for the purpose of the generalised Hawk T.Mk1, any sub-structures which are mirrored on either side of the fuselage are stated to be a mirrored copy of its sibling sub-structure. To this end, the work outlined within this chapter will focus on the right side of the plane and assert that the left counterparts are mirrored sub-structures: i.e. the left wing is a mirrored sub-structure of the right wing.

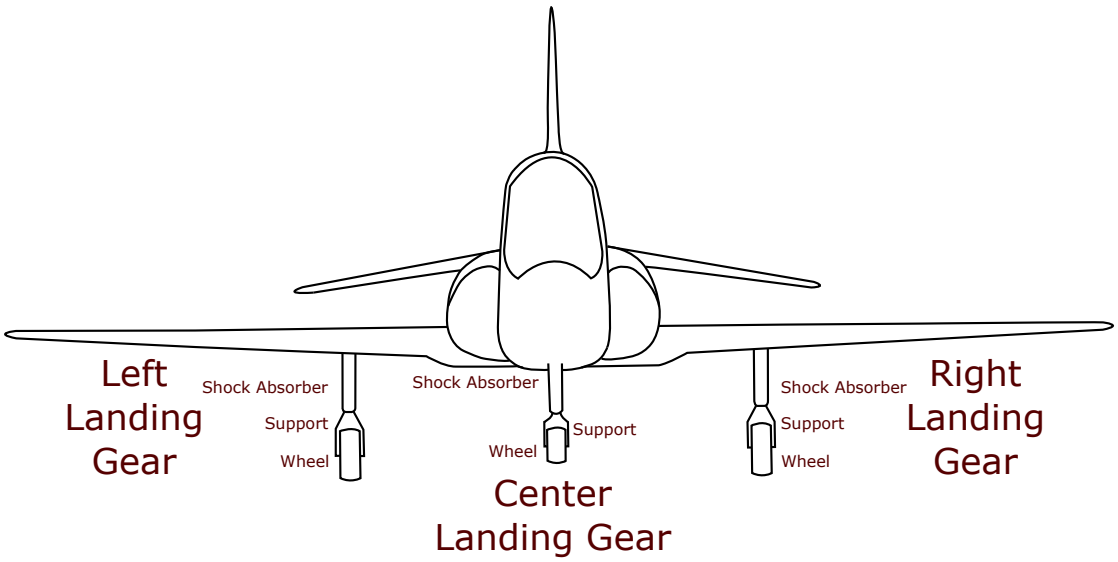
If one first considers the fuselage and vertical stabilisers, Figure 6.7a shows the breakdown of the two sub-structures into the respective boundaries for [regular] *element* curation. The divisions noted for the fuselage can be directly related to

either geometrical changes on the fuselage — for instance, where the fuselage has a distinct geometric change to house the canopy for the pilot — or where the fuselage has visible manufacturing contours present — see Figure 6.1.

The minimum required knowledge for the curation of a [regular] *element* is *contextual* type, *material* type, and *geometry* type. Whilst the exact *material* composition of the airframe is unknown at the time of writing, for the purposes of an IE model, one could make the assumption that the material used on the shell is a derivative of aluminium: as such, 'metal' →'aluminiumAlloy' has been selected for inclusion within the model. While one could use the standard geometrical types available



(a) The side profile of the Hawk T.Mk1 depicting the segmentation of the fuselage and vertical stabiliser.



(b) The nose view of the Hawk T.Mk1 depicting the segmentation of the landing gear.

Figure 6.7: The segmentation on the Hawk T.Mk1 into [regular] *element*s for the fuselage, vertical stabiliser and landing gear.

within the IE model language to describe the aeroplane, any visualisations of the models' form would not accurately reflect the form of the Hawk T.Mk1; instead, the advanced geometrical type of 'shell' →'translateAndScale' →'cylinder' is used within the model.

6.3.1 Geometrical data method

Whilst the dimensions of the *geometry* object are not required for the generation of a valid IE model, they are potentially advantageous when considering the evaluation of similarity within the *network*. As such, a method was developed for capturing the geometrical dimensions on such a large object. If one considers the marked boundaries for [regular] *element*s depicted in Figure 6.7a as a *face* within a [cuboid] *bounding box*, one needs to capture the x and y movements within the *face* to encapsulate the shape.

The method used, was in the direction of the *face*, provide two *fixed reference*s (k_t and k_b) which the shape within the *face* would be measured against. Figure 6.8

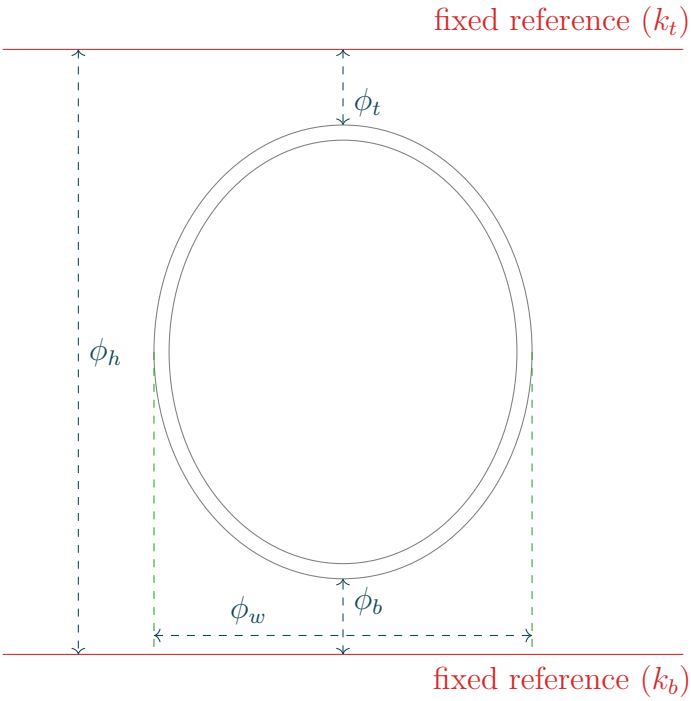


Figure 6.8: The measurements taken of each *face* for the fuselage of the Hawk T.Mk1.

shows the example of such an aforementioned shape and the relation between the two fixed references. The first measurement (ϕ_h) measures the distance between the two fixed references. The second measurement (ϕ_t) is from the top of the shape within the *face* and top fixed reference (k_t). The third measurement (ϕ_b) is from the bottom of the shape within the *face* and the bottom fixed reference (k_b). The fourth and final measurement is between the left and right furthermost edges of the shape to get the width of the shape (ϕ_w).

To acquire these dimensions for the real-world Hawk in the Laboratory for Verification and Validation (LVV), two fixed references were used: a crane locked out at a fixed height attached to a flat surface (k_t), and the floor of the LVV (k_b). Figure 6.9 shows the modifications made to the crane to provide the fixed reference k_t. Figure 6.10b and Figure 6.10c show the laser measure used to measure the distance between two points. To measure the width of the shape at the given *face* (ϕ_w), a plumb line was used to transfer the point onto the plane of k_b (see Figure 6.10a). Once both points from the airframe were transferred to k_b, the distance

Figure 6.9: The top fixed reference used for the measurements of the Hawk T.Mk1: a crane locked at a fixed height in the Laboratory for Verification and Validation, thus facilitating the fixed reference to be moved to the desired position on the aeroplane.

between these points was measure to give ϕ_w. These measured values can then be transformed into the dimensions supported within 'shell' →'translateAndScale' →'cylinder'.

(a) A plumb line was utilised to transfer a position on the aeroplane to the bottom fixed reference — the floor of the Laboratory for Verification and Validation.

(b) A laser distance measure was utilised to measure points on the hawk to either reference object.

(c) The laser distance measure was placed in a jig to utilise gravity to ensure the distance measure was as square as possible to the reference object when the surface was not horizontal.

Figure 6.10: The equipment used to generate the measurement dimensions set forth in Figure 6.8.

6.3.2 Elements and relationships

To depict the segmentation used on the generalised Hawk T.Mk1, three figures have been provided to note the approximate boundaries between [regular] *elements*. Figure 6.7a shows the side view of the Hawk T.Mk1 and depicts the segmentation along the fuselage and vertical stabiliser. As aforementioned, the segmentation along the fuselage has occurred at any instance where large geometrical changes arise within the fuselage or when damage detection is desired. For instance, towards the end of the fuselage, it is beneficial to know where, within the fuselage, the interactions between the vertical and horizontal stabilisers take place, as such, the fuselage is segmented to facilitate this desired localisation within the model.

Figure 6.7b shows the view of the Hawk from the nose of the aeroplane to depict the segmentation in the landing gear. For the purpose of generating the IE model, the Hawk is modelled in the situation that the real-world example is found within the LVV: at rest on the ground being supported through its landing gear. The desire behind the Hawk IE model, is to serve as a basis for finding similarity throughout the airframe of an aeroplane; as such, the landing gear has been simplified to only the shock absorber, support structure and wheel, as it is important to provide the context as to how the model is supported from the ground, but is not required for airframe matching.

Figure 6.11 shows the final perspective of the Hawk T.Mk1, from directly above the aircraft, to provide the segmentation for the wings and horizontal stabilisers. For both the wings and the horizontal stabilisers, the segmentation of [regular] *element*s has occurred along features which are present upon the real-world aircraft. Because of the movement required within a wing to provide the functionality of the flaps, they are not often of a singular construction. There are gaps present to provide a clearance for the flap and aileron movement within the wing. In the case of the flaps and the ailerons, these gaps provide a natural boundary for the corresponding [regular] *element*s; however, these boundaries can then be extended to provide segmentation for the other sections of the wing. Figure 6.2 shows how the boundaries for the flaps and the ailerons have been extended on the real-world Hawk to provide a damage localisation-based segmentation for the rest of the wing — the extension of the segmentation is depicted by a blue chalk line on the wing. The outline of the Hawk used within Figures 6.7a, 6.7b, and 6.11, is based upon a simplified tracing of the drawings constructed by Rolfe [100].

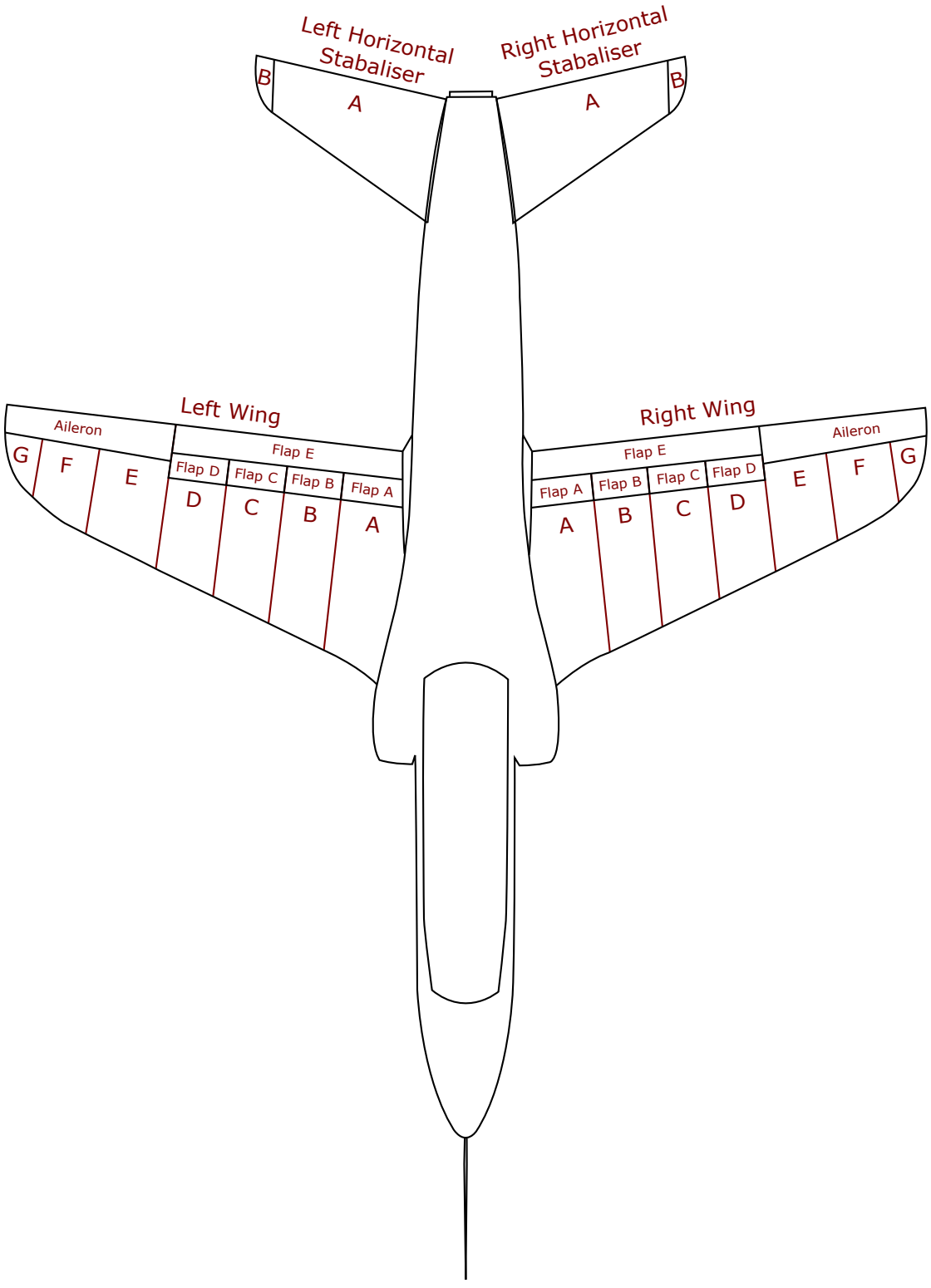


Figure 6.11: The segmentation on the Hawk T.Mk1 into [regular] *element*s for the left wing, right wing, left horizontal stabiliser, and right horizontal stabiliser.

For brevity, the full tables including all *element* and *relationship* attributes are not included within this thesis; instead, an abridged version is included with the minimum required information for each *root object* as per the PBSHM Schema v1.1.0 specification included within this thesis. Table 6.1 lists the [regular] *element*s curated using the segmentation described in Figures 6.7a and 6.11 for the fuselage and right-wing sub-structures. In the interests of chapter narrative, the tables for the other *element*s curated during the segmentation depicted in Figures 6.7a, 6.7b, and 6.11 are excluded from the main body of this chapter and instead included in the Appendix. The left wing with the vertical and horizontal stabilisers is covered in Table B.1, the landing gear in Table B.2, and the [ground] *element*s in Table B.3.

Fuselage

Element Names	Type Attributes
antenna	Contextual: 'other' Material: 'metal' →'aluminiumAlloy' Geometry: 'solid' →'translateAndScale' →'cylinder'
fuselage-a, fuselage-b, fuselage-c, fuselage-d, fuselage-e, fuselage-f, fuselage-g, fuselage-h, fuselage-i, fuselage-j, fuselage-k, fuselage-l	Contextual: 'fuselage' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'

Right Wing

Element Names	Type Attributes
right-wing-a, right-wing-b, right-wing-c, right-wing-d, right-wing-e, right-wing-f, right-wing-g	Contextual: 'wing' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'
right-flap-a, right-flap-b, right-flap-c, right-flap-d, right-flap-e	Contextual: 'aerofoil' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'
right-aileron	Contextual: 'aerofoil' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'

Table 6.1: The [regular] *element*s for the Fuselage and Vertical Stabiliser of the generalised Hawk T.Mk1.

Using the rules defined within this thesis for *relationship* types, [regular] *element*s that have been created to segment a larger *element* are represented by [perfect] *relationship*s, any presence of movement between [regular] *element*s is represented

by [joint] *relationship*s with a [dynamic] *nature*, any other interactions between [regular] *element*s are modelled via [joint] *relationship*s with a [static] *nature*; finally, any interactions with [ground] *element*s are modelled using [boundary] *relationship*s.

Fuselage

Relationship Name	Element Set	Type
antenna-fuselage-a	{antenna, fuselage-a}	[joint], [static]
fuselage-a-b	{fuselage-a, fuselage-b}	[perfect]
fuselage-b-c	{fuselage-b, fuselage-c}	[perfect]
fuselage-c-d	{fuselage-c, fuselage-d}	[perfect]
fuselage-d-e	{fuselage-d, fuselage-e}	[perfect]
fuselage-e-f	{fuselage-e, fuselage-f}	[perfect]
fuselage-f-g	{fuselage-f, fuselage-g}	[perfect]
fuselage-g-h	{fuselage-g, fuselage-h}	[perfect]
fuselage-h-i	{fuselage-h, fuselage-i}	[perfect]
fuselage-i-j	{fuselage-i, fuselage-j}	[perfect]
fuselage-j-k	{fuselage-j, fuselage-k}	[perfect]
fuselage-k-l	{fuselage-k, fuselage-l}	[perfect]

Right Wing

Relationship Name	Element Set	Type
right-wing-a-b	{right-wing-a, right-wing-b}	[perfect]
right-wing-b-c	{right-wing-b, right-wing-c}	[perfect]
right-wing-c-d	{right-wing-c, right-wing-d}	[perfect]
right-wing-d-e	{right-wing-d, right-wing-e}	[perfect]
right-wing-e-f	{right-wing-e, right-wing-f}	[perfect]
right-wing-f-g	{right-wing-f, right-wing-g}	[perfect]
right-wing-flap-a	{right-wing-a, right-flap-a}	[joint], [dynamic]
right-wing-flap-b	{right-wing-b, right-flap-b}	[joint], [dynamic]
right-wing-flap-c	{right-wing-c, right-flap-c}	[joint], [dynamic]
right-wing-flap-d	{right-wing-d, right-flap-d}	[joint], [dynamic]
right-wing-e-aileron	{right-wing-e, right-aileron}	[joint], [dynamic]
right-wing-f-aileron	{right-wing-f, right-aileron}	[joint], [dynamic]
right-wing-g-aileron	{right-wing-g, right-aileron}	[joint], [dynamic]
right-wing-a-flap-e	{right-wing-a, right-flap-e}	[joint], [dynamic]
right-wing-b-flap-e	{right-wing-b, right-flap-e}	[joint], [dynamic]
right-wing-c-flap-e	{right-wing-c, right-flap-e}	[joint], [dynamic]
right-wing-d-flap-e	{right-wing-d, right-flap-e}	[joint], [dynamic]
right-wing-a-fuselage-f	{right-wing-a, fuselage-f}	[joint], [static]

Table 6.2: The *relationship*s for the Fuselage and Vertical Stabiliser of the generalised Hawk T.Mk1.

The *relationship*s defined via the segmentation present within the fuselage and

right wing — as per the Figures 6.7a and 6.11 — are included within Table 6.2. As discussed previously, the *relationship*s between the other *element*s defined in Figures 6.7a, 6.7b, and 6.11 are included within the Appendix of this thesis. The left wing and vertical stabiliser is covered in Table B.4, the vertical and horizontal stabilisers to the fuselage *relationship*s are covered in Table B.5, and the landing gear *relationship*s are covered in Table B.6.

6.3.3 Graphs

Using the graph notation introduced within Section 6.2, the IE model representation of the generalised Hawk T.Mk1 is transformed to its final resting place in the form of an AG. The AG notation not only provides a concise methodology for communicating the web of interconnections between the *element*s and *relationship*s within a model, but provides a visual representation of the form that a structure takes.

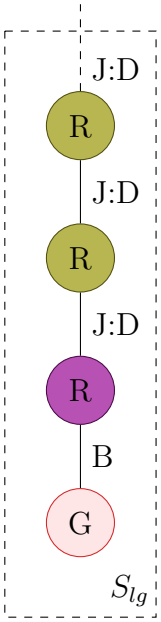


Figure 6.12: The subgraph for the landing gear in the generalised Hawk T.Mk1.

The standard notation and labelling set forth in Section 6.2 is expanded for the purpose of the generalised Hawk T.Mk1 graphs to include a colour scheme for representing the *contextual* type within a [regular] *element*. A *contextual* type of 'fuselage' is represented via a purple filled vertex, a *contextual* type of 'aerofoil' is

represented by a green filled vertex, a *contextual* type of 'wing' is represented by a blue filled vertex, a *contextual* type of 'wheel' is represented by a pink filled vertex, and a *contextual* type of 'other' is represented by an olive filled vertex.

To aid in the visual representation of the Hawk T.Mk1, the graph has been split into two views: the 'top down' view (see Figure 6.14), and the 'side' view (see Figure 6.13). The following subgraphs are present within the model: the landing gear (S_{lg}), the right wing (S_{rw}), and the right horizontal stabiliser (S_{rhs}).

Figure 6.12 depicts the subgraph of the landing gear: S_{lg}. As discussed previously, the landing gear — for the purpose of the model in question — has been simplified in nature, and as such, is declared as a subgraph — external to the main graphs — which can then be placed within the main body of the graph via a reference node.

Within the 'side' view of the Hawk graph (see Figure 6.13), the wings and horizontal stabilisers are omitted from the graph to align the graph to the displayed segmentation in Figure 6.7a. There are no subgraphs declared within Figure 6.13; however, the landing gear subgraph S_{lg} is referenced to depict the nose landing gear location within the fuselage.

Within the 'top down' view of the Hawk graph (see Figure 6.14), the subgraphs S_{rw}

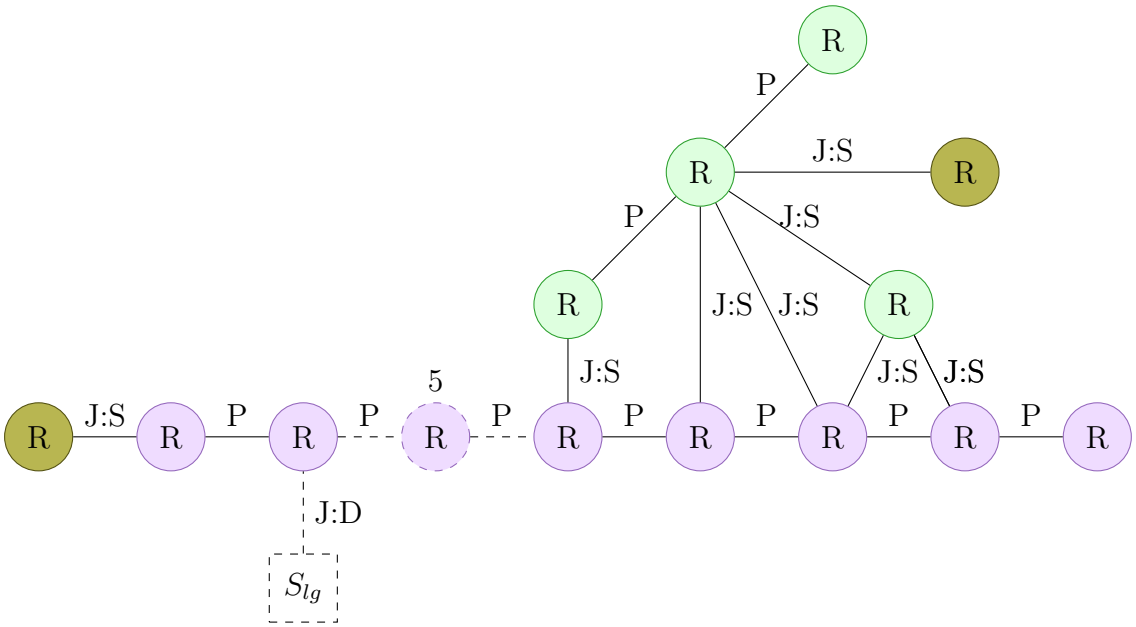


Figure 6.13: The graph of the 'side' view of the generalised Hawk T.Mk1 representing the fuselage and vertical stabiliser.

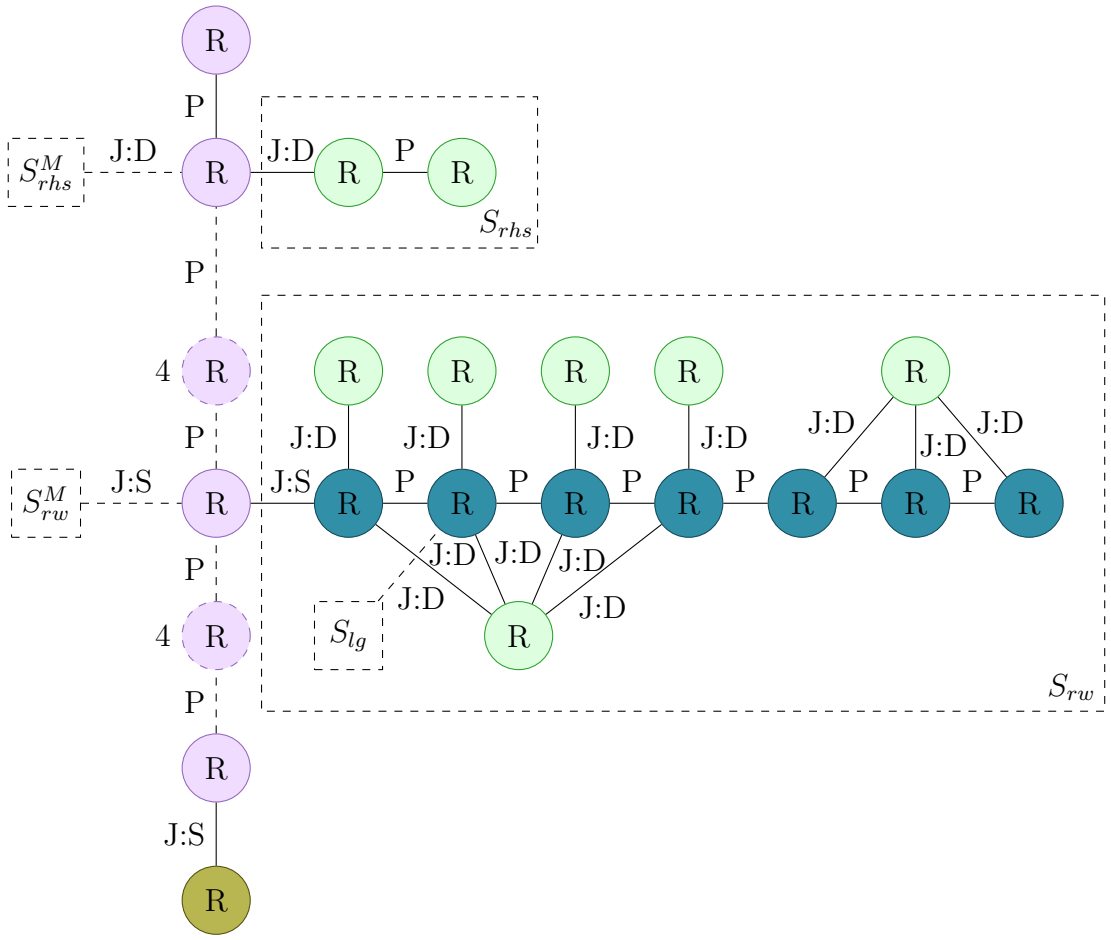


Figure 6.14: The graph equivalent to Figure 6.14 of the generalised Hawk T.Mk1.

and S_{rhs} are declared, with a mirrored reference of the graphs S^M_{rw} and S^M_{rhs} also present within the graph on the opposite side of the fuselage vertices. As part of the subgraph S_{rw}, the subgraph S_{lg} is also referenced; thus, when a reference to S_{rw} is made, the S_{lg} subgraph is also included.

6.4 Conclusion

In conclusion, this chapter demonstrates that IE models are not just for civil structures and can be used across a range of structure types — including aerospace; however, the methodology utilised within the curation of IE models will require adaptation and modification dependent upon the structures in question.

A new graph notation has been introduced to simplify the associated graphs of real-

world structures via the use of subgraphs, reference nodes and mirrored reference nodes. A methodology was introduced for measuring aerospace structures and breaking down these large complex structures into repeatable steps for capturing their geometric representations.

Finally, the introduced IE model language, AG notation and measuring techniques were applied to a real-world aircraft in the Laboratory for Verification and Validation; a Hawk T.Mk1.

Chapter 7

Conclusions and Future Work

This thesis has focussed on the issues faced within the similarity assessment portion of PBSHM. The goal of PBSHM is to be able to transfer learnt knowledge from a source structure to a target structure via a process called transfer learning; however, with the aim of reducing the likelihood of negative transfer — when the transferred knowledge negatively impacts the knowledge of the target structure — PBSHM first aims to establish a base similarity between a group of structures — the *population* — to guide if transfer learning should be attempted.

To facilitate the aforementioned similarity between structure, one needs to establish a shared domain in which all structures may be compared equally. The first section of this thesis radically reconstructed the core concepts, definitions, and language of an Irreducible Element (IE) model to enable embedding additional engineering knowledge and design choices within the model. The second section of this thesis introduced a novel ecosystem and shared-domain context for PBSHM similarity computations: *network*, *framework*, and *database*. The third section of this thesis looked at the problems present during similarity comparisons because of author bias, and how the *network* can adapt to this bias, via the reconstructed language of an IE model and the use of the *Canonical Form*. Lastly, a case study of generating an IE model on a real-world structure is given via the Hawk aeroplane and a new graphical notation for IE models is introduced.

Chapter 3 introduced the concept of an Irreducible Element (IE) model. Before any attempt can occur to determine the similarity between two structures, there

first must be a common reference for describing the structures in question, to facilitate a level playing field. The IE model is the aforementioned vehicle in PBSHM to describe structures; however, the knowledge contained within this domain — prior to this thesis — did not facilitate either a standardised language for representation within an algorithmic space or encapsulate enough knowledge regarding the engineering knowledge and design choices for the *network* to reliably understand certain modelling scenarios. Therefore, a radical reconstruction of both the IE model language and the knowledge contained within the domain was required.

The reconstructed language within the IE model domain now unifies any physical component within a model as an *element* and any interaction between these components as a *relationship*. Different use-case scenarios of an *element* or *relationship* are embedded within the language via the use of *object types*. [regular] *element*s represent a component that is part of the structure being modelled, and a [ground] *element* represents any component that should be considered as external to the current structure being modelled. [perfect] *relationship*s represent when the two [regular] *element*s should be considered as one larger [regular] *element* and have only been divided to encapsulate a components' complex geometrical form. A [connection] *relationship* is where two or more [regular] *element*s are held together by an unknown component excluded from the model; a [joint] *relationship* is where the physics between two [regular] *element*s is desired to be modelled, and a [boundary] *relationship* denotes where a [regular] *element* comes in contact with a [ground] *element* and indicates the boundary of the structure in question.

Furthermore, the model itself may also be determined as a [free] *model*, where the structure is considered in a void — no [ground] *element*s or [boundary] *relationship*s may be present within the model — or as a [grounded] *model* where the interactions between the structure and any external systems are included in the model. The language of an IE model is now also defined via a standardised representation, providing each *object* with a separated subdomain of accepted attributes and values to facilitate an equal description of structural properties and knowledge across the *network*.

Chapter 4 introduced the holistic ecosystem to provide the technical grounding for PBSHM technology. Akin to the work conducted in Chapter 3, Chapter 4 examined the requirements for a technical ecosystem to provide a shared domain for PBSHM technology. Because of the nature of PBSHM, data are required to be processed

on multiple structures and datasets. To aid in the adoption of PBSHM technology, a shared computational space is desired to operate universally across all structures and associated data that may be present within PBSHM; as such, three shared data domains were identified as being necessitated: data domain (the *database*), computational domain (the *framework*), and the similarity domain (the *network*).

The *database* was implemented via a NoSQL database — MongoDB to be precise — and a PBSHM-specific schema was built to regulate and accept data in a shared format. The database over time will need to accept all relevant PBSHM data; however, for the purpose of this thesis, the focus was placed upon implementing and supporting channel data and IE model data. Channel data were defined as any raw data occurring from a sensor on a structure from either a long-term monitoring campaign or from a short-term experimental setup. The standardised representation of IE model data defined in Chapter 3 is implemented into the PBSHM schema via the native JSON document format within MongoDB.

The *framework* was implemented via a Python web framework — based upon the micro-web framework: *flask*. Rather than the framework being a single entity, it is instead split into a core computational platform, whose functionality can then be extended via modules. This scenario not only enables each module to support and govern its own domain of the PBSHM problem, but permits algorithms to be built in conjunction with research developments. Akin to the database, the framework over time will need modules curated that support the full spectrum of computational algorithms necessitated in PBSHM; however, for the purpose of this thesis two modules were implemented inline with the supported data. The first module enables the viewing of channel data stored within the database. The second module facilitates the generation of similarity metrics via the Maximum Common Subgraph (MCS) and Jaccard Index — again, data is pulled directly from the database. The Jaccard Index is then calculated for an example eight-bridge dataset natively within the framework, showcasing how similarity metrics in the *network* can now be generated using the reconstructed language of IE models, the database, and the framework.

The *network* is the space in which the connections between structures live. Whilst the database and the framework may provide the shared domain and technical implementations for both the data and computations to respectfully reside, the network is the shared domain where all of these individual items come together to build the *network of similarity*. The definition of the network is expanded to include

not only IE model data and similarity metrics, but to include any data contained within the database or framework. The network of similarity is a fully-connected graph where each node in the graph is a structure within PBSHM, any available data on that structure is then a sub-node of the structure node. The similarities between structures are stored as edges within the network; therefore, because of potentially multiple similarity metrics within PBSHM, each structure may have multiple edges to another node in the network.

Chapter 5 introduced the problem of variations being present within an IE model, as a result of the inherent issue of author bias during the curation of IE models from real-world structures. Nuanced variations within a model, better reflect the current understanding and knowledge of the author on the structure; however, these same variations in models, provide a computational conundrum when determining the similarity. If the *network* cannot identify when two models are from the same structure and therefore identical, the premise of generating a similarity between different structures falls short. As such, for comparison within the *network*, a single structure should have only one unique representation within the network; regardless of any human-induced variations.

The *Canonical Form* IE model is introduced as the vehicle for the aforementioned single representation per unique structure. A *detailed* IE model — a model generated by a human — is reduced down to the Canonical Form version via the *Canonical Form Reduction Rules* (CFRR). The reconstructed language of an IE model, enables the CFRR to understand why certain variations within the attributed-graph representation are present, and ultimately enables the variations to be removed from the graph, without invalidating any knowledge which is later relied upon within the *network of similarity*. The chapter introduces the first three CFRR: unique [ground] *element*s, [perfect]-[joint]-[joint] *relationship*s, and [perfect] *relationship*s. The concept of a *Reality Model* is introduced within the chapter. An IE model only contains knowledge pertinent to the composition of a structure. The Reality Model contains all the information regarding the world in which the IE model is placed. Whilst the notation of a Reality Model is not expanded upon in this thesis, it is important to understand the role in which the Reality Model may affect the potential reductions made, to derive the Canonical Form representation.

The purpose of the Canonical Form is thus expanded to directly interact within the network. In the network, instead of each structure requiring similarity metrics calculated to each other structure, the Canonical Form is proposed as a known

reference within the network, to reduce the number of computations necessitated in the network. Each structure would have the similarity metrics calculated to only the known Canonical Form reference models, instead of every structure within the network.

The case study of a simplified beam and slab bridge is introduced. The Canonical Form representations are generated for the simplified beam and slab bridges from two to ten spans. An input dataset is also generated for the simplified beam and slab bridges from two to ten spans; however, these input bridges contain the highlighted variations within the graphs. Once again, the MCS and Jaccard Index is utilised to generate two similarity matrices. The first matrix contains the similarity scores when comparing the input graphs against the known representation graphs without any reduction of the input graphs. The second matrix performs the same similarity comparison as the first matrix; however, the input graphs are reduced via the CFRR before comparison to the known representations. The results highlight that without the CFRR, the algorithm in question is not able to find any noticeable pattern of similarity within the two datasets; however, when the CFRR are used, the algorithm is able to find the start of a pattern of similarity in the two datasets. This work highlights the importance of the discovery and development of the CFRR within PBSHM.

This Chapter also introduces the idea of using an algorithm from a Machine-Learning (ML) domain instead of a graph-theory domain, to compute the similarity in the network. The algorithm in question is a Graph Matching Network (GMN) algorithm, where either pairs or triplets of graphs can be utilised to teach the algorithm what is similar or not. The GMN algorithm is utilised to generate the same similarity matrix as used in the MCS and Jaccard Index results. The GMN is successfully able to identify the desired pattern of similarity between the input graphs and the known representations graphs.

Chapter 6 demonstrated that IE models are not restricted to civil engineering structures and are applicable to a full range of real-world structures; namely in this chapter, aerospace structures. The generalised Hawk T.Mk1 is introduced as a real-world aeroplane structures for which an IE model is generated. Whilst the rules for what constitutes an *element* and *relationship* remain constant across all structures, the methodology utilised to break down the structure into the individual components for inclusion with the model, must be adapted and modified dependent upon the class of structure being examined. For the generalised Hawk T.Mk1, the

chapter depicted how to break down the airframe into an IE model and the process used to translate the geometrical representation of the real-world structure in the LVV, to the attributes available within the IE model language.

In PBSHM, representing an IE model of a real-world structures in a graph visualisation can often prove a complicated endeavour, because of the 3D nature and scale of the graph. To facilitate concise visual representation of these graphs, PBSHM required a new notation for the graphs derived from real-world IE models. The aforementioned notation is introduced in Chapter 6 to minimise the repetition within the graph via the inclusion of subgraphs, reference nodes and mirrored reference nodes. The graph of the generalised Hawk T.Mk1 is included within the Chapter.

7.1 Future Work

The work outlined in this thesis provides the first holistic computational conceptualisation for PBSHM. Whilst this work may be the first of its kind in the field, there is still further work to be completed on each component of the technological ecosystem described within this thesis, to achieve the lofty objectives of PBSHM.

The reconstructed language of IE models, makes significant progress towards achieving the end purpose of the model: being able to embed the structural knowledge and composition of any structure, regardless of type, within PBSHM. There are however, gaps within the language which prevent a full embedding of engineering knowledge on certain classes of structures. Currently, an IE model has the language to describe the geometrical properties for an I-beam; however, there are other beam-related geometrical representations which require supporting within an IE model: C-beam, H-beam, W-beam, etc. The geometrical representations of a [regular] *element* as a whole, will need further expansion to embed yet unseen forms of components within the model.

There are also gaps within the theory of an IE model which themselves will also require addressing. There is currently no vehicle within an IE model for the modeller to inform the *framework* and subsequently the *network* regarding the granularity of the model. Has the modeller focussed in great detail on modelling

the interaction between a specific component, whilst giving a very abstract and general representation for the rest of the model? Akin to the understanding that the CFRR provide the *network* regarding potential variations, understanding the granularity of a model — or subsection of a model — may prove invaluable for generating similarity metrics across the range of varied structures within PBSHM.

Another IE model theory which may require potential expansion over time, is the reference to external systems within a model. Currently, the vehicle for achieving this is a [ground] *element*; however, there is no reference included as to what the external system in question is. As the external system could also be modelled as an IE model, it is envisioned that the definition of a [ground] *element* may be expanded upon in the future to include a reference to another IE model, if such a model exists. This approach would bolster the *network*'s understanding of the context of any directly-related structures. This same approach may also be included to achieve a referencing of common subsections of an IE model. If one pictures the Hawk T.Mk1 included in Chapter 6, within the *network* there may be a detailed IE model (an IE model with a high level of granularity) of a particular subsection of the aeroplane; say the landing gear. The landing gear may be used across multiple makes and models of aeroplane, so instead of requiring the modeller to encapsulate the intricate workings of the components which form the landing gear, they instead can reference an external IE model which contains the relevant information. The external referencing principle and the proposed granularity labels could also then be combined for referencing generic high-level sections of a structure, as the purpose for why the IE model is being generated may only be concerning a certain section of the structure and not the high-level generic sections.

The *Reality Model* was introduced within the thesis as the vehicle for embedding knowledge regarding the world in which the IE model lives in PBSHM. It is fair to say that the Reality model is currently just a notion of what it may eventually be. The desire is that the Reality model will encapsulate all currently available knowledge on a structure. While the structural component of the Reality model will be delegated to the IE model definitions and language, there are vast areas of knowledge which require development of their own definitions and language to correctly embed their knowledge within the model. Four of the aforementioned key knowledge areas are: Environmental and Operational Variables (EOV's), features, labels, and sensor placement. Enabling the embedding of these key knowledge areas into the Reality Model, is crucial for facilitating SHM-driven problems within PBSHM. These areas

of research into the Reality model, will directly lead into future developments in the *network*, *framework*, and *database*. As additional areas of knowledge are supported within a Reality model, the *network*, *framework*, and *database* must be updated to include these areas of knowledge within their associated shared data domain.

Removing unrequired variations from models within the *network of similarity* is going to be a key area of research going forward, if PBSHM is ever going to realise its objective. The *Canonical Form* is the vehicle within PBSHM, to have a network free of variations which will inevitably hamper potential population generation via similarity metrics. The results of the CFRR verses the GMN in Chapter 5 highlight two key points for future research. The first core area of research needs to be into the CFRR. The CFRR are clearly lacking total coverage of all potential reductions of variations, even in a simplified simulated dataset. Further work needs to be conducted into expanding the current three derived rules and potential new rules of reduction or inclusion: it is envisioned that the CFRR in the future may not only remove *element*s from the model, but may also include missing *element*s from a model.

As the CFRR are developed to understand what data are required within the network, undoubtably the version of what is stated to be the Canonical Form will also change. The evolving reference is not problematic, as IE models will be stored within the database as their *detailed* author-submitted versions. New Canonical Form version of the detailed models will be generated inline with the evolving knowledge around network requirements. The Canonical Form produced today from a detailed model, is highly unlikely to be the Canonical Form produced in the future from the same detailed model.

The second highlighted area of further research from the similarity comparisons is that of utilising an ML approach within the network. The GMN result highlighted the power of utilising an ML approach to similarity as the GMN was able to find patterns and reductions not known to the CFRR; however, an ML approach is not a panacea solution for similarity. To teach the GMN, it required a not insurmountable amount of data to train the algorithm for successful recognition of the variations present within the dataset. Whilst this is fairly easy to overcome when utilising a simulated dataset, in the real-world, there are not enough IE models of real structures yet, to accomplish such an ML-based approach. As with most things in life, one needs to explore both paths on the road to similarity in PBSHM. Whilst PBSHM is in its infancy, one will need to rely on non-ML approaches, such as the

MCS from graph theory to address the similarity problem. This issue subsequently necessitates the highlighted research opportunities with the CFRR. When PBSHM has advanced in years, and has a greater amount of structures modelled within the network, ML approaches may prove to be the favoured solution because of thier ability to showcase yet unknown relations.

Appendix A

PBSHM SCHEMA

A.1 Root Schema

Property	Description	Type
version	The version of the PBSHM Schema that the document is compliant towards. Accepted values: '1.1.0'	*string*
name	A unique name of the structure within the population.	*string*
population	The name of the initial type of structures to which the structure belongs.	*string*
timestamp	Timestamp of when the associated data were recorded, stored in UTC nanoseconds since UNIX epoch.	*long*
models	Associated models for the given structure. See Table A.2 for the available child properties.	*object*
channels	Associated raw channel data acquired from either a short or long-term monitoring campaign. See Table A.39 for the available child properties and see Table A.40 for a list of accepted values.	

Required Properties: ***version***, ***name***, ***population*** and ***timestamp***

Table A.1: The available properties for the declaration of a structure within the PBSHM Schema.

Property	Description	Type
irreducibleElement	The Irreducible Element (IE) model that describes the structure at the given time point. See Table A.3 for the available child properties.	*object*

Required Properties: None

Table A.2: The available properties for the declaration of a model within the PBSHM schema.

A.2 IE Model Schema

Property	Description	Type
type	The selected type of the *model*, either 'free' for a [free] *model* or 'grounded' for a [grounded] *model*.	*string*
elements	An array of *element* objects. See Figure 3.5 for the available *element*s dependent upon the selected *model* type.	*array*
relationships	An array of *relationship* objects. See Figure 3.5 for the available *relationship*s dependent upon the selected *model* type.	*array*

Required Properties: ***type*** and either ***elements*** or ***relationships***

Table A.3: The available properties for the declaration of an Irreducible Element model within the PBSHM schema.

A.2.1 Regular element

Property	Description	Type
name	The unique name for the *element* within the *model*. Must have a length between 1 and 64 characters.	*string*
description	Additional none structured information to describe the *element*.	*string*
type	Selected type of *element*. Accepted values: 'regular'.	*string*
coordinates	Position of the [regular] *element* within the *model*s coordinate space. See Table A.5 for the available child properties.	*object*
contextual	Purpose of the [regular] *element* has within the *model*. See Table A.7 for the available child properties.	*object*
geometry	Details of the shape and volume that the [regular] *element* ocupies within the *model*s coordinate space. See Table A.8 for the available child properties.	*object*
material	Details of the material and associated properties that the [regular] *element* is constructed from. See Table A.15 for the available child properties.	*object*

Required Properties: ***name***, ***type***, ***contextual***, ***geometry*** and ***material***

Table A.4: The available properties for the declaration of a [regular] *element* within the PBSHM schema.

Coordinates

Property	Description	Type
global	Translational and Rotational coordinates within the global coordinate space. See Table A.6 for the available child properties.	*object*

Required Properties: ***global***

Table A.5: The available properties for the coordinate object within a [regular] *element* in the PBSHM schema.

Property	Description	Type
translational	X, Y and Z translational values within the global coordinate space. See 'Translational coordinates Object' in Table A.35 for the available child properties.	*object*
rotational	Alpha, Beta and Gamma rotational values within the global coordinate space. See 'Rotational Coordinates Object' in Table A.35 for the available child properties.	*object*

Required Properties: ***translational***

Table A.6: The available properties for the global coordinate object within a [regular] *element* in the PBSHM schema.

Contextual

Property	Description	Type
type	The type that describes the purpose of the *element* within the *model*. Accepted values: 'wall', 'slab', 'beam', 'cable', 'block', 'plate', 'column', 'deck', 'aerofoil', 'wing', 'fuselage', 'tower', 'wheel' or 'other'.	*string*

Required Properties: ***type***

Table A.7: The available properties for the contextual object within a [regular] *element* in the PBSHM schema.

Geometry

Property	Description	Type
type	The shape that describes the volume which the *element* occupies. See Table A.34 for the available nested child properties. See Table A.9 for the list of accepted standard types and Table A.14 for the list of accepted complex types.	*object*
bounding	Dimensions of the smallest box which encapsulates all the associated dimensions of the *element*. See Table A.10 for a list of child properties.	*object*
faces	The dimensions and properties of each face of the complex shape being described. See Table A.11 for a list of child properties. Only available when the shape is a complex geometry (See Table A.14).	*object*
dimensions	Each measurable dimension associated with the shape being described which belong to the whole shape. There is one child property for each given dimension. Child properties are either a named property (See Tables A.9 and A.14), or they are wildcard properties which can have any valid JSON string as their name. The value for the property is an *object* for which the child properties can be found in Table A.38. For a wildcard property, the accepted values are 'Wildcard' under 'Value Object Type'. For named properties, the accepted value type are declared in Table A.9 for standard geometrical types and in Table A.14 for complex geometrical types.	*object*

Required Properties: ***type***. For any complex types declared in Table A.14: ***bounding*** if ***faces*** or ***dimensions*** declared, ***faces*** if ***bounding*** or ***dimensions*** declared.

Table A.8: The available properties for the geometry object within a [regular] *element* in the PBSHM schema.

Type Tree	Dimension Property	Dimension Object
beam* → *rectangular	***length***, ***width***, ***height***	*Linear*
beam* → *circular	***length***, ***radius***	*Linear*
beam* → *i-beam	***length***, ***d***, ***h***, ***s***, ***b***, ***t***	*Linear*
beam* → *other	***length***	*Linear*
plate* → *rectangular	***thickness***, ***width***, ***length***	*Linear*
plate* → *circular	***thickness***, ***radius***	*Linear*
plate* → *other	***thickness***	*Linear*
solid* → *translate* → *cuboid	***length***, ***width***, ***height***	*Linear*
solid* → *translate* → *sphere	***radius***	*Linear*
solid* → *translate* → *cylinder	***radius***, ***length***	*Linear*
solid* → *translate* → *other		
shell* → *translate* → *cuboid	***thickness***, ***length***, ***width***, ***height***	*Linear*
shell* → *translate* → *sphere	***thickness***, ***radius***	*Linear*
shell* → *translate* → *cylinder	***thickness***, ***radius***, ***length***	*Linear*
shell* → *translate* → *other	***thickness***	*Linear*

Required Properties: Any named dimension property in the Table above. See Table A.38 for the available child properties and, within the table, 'Linear' under 'Dimension Object Type' for the accepted values.

Additional Information: Additional dimensions can be declared as a *Wildcard*. See Table A.38 for the available child properties and, within the table, 'Wildcard' under 'Dimension Object Type' for the accepted values.

Table A.9: The available standard geometrical types and their associated required dimensions properties and accepted object types for a [regular] *element* in the PBSHM schema.

Property	Description	Type
type	The type of shape used to encapsulate the *element*. Accepted Values: 'cuboid'.	*string*
length	The measured length of the cuboid bounding box. See Table A.38 for the available child properties and, within the table, 'Linear' under 'Dimension Object Type' for the accepted values.	*object*
width	The measured width of the cuboid bounding box. See Table A.38 for the available child properties and, within the table, 'Linear' under 'Dimension Object Type' for the accepted values.	*object*
height	The measured height of the cuboid bounding box. See Table A.38 for the available child properties and, within the table, 'Linear' under 'Dimension Object Type' for the accepted values.	*object*

Required Properties: ***type***, ***length***, ***width*** and ***height***

Table A.10: The available properties for the cuboid bounding box object within a [regular] *element* in the PBSHM schema.

Property	Description	Type
left	The left face of the shape within the bounding box; the face at which the value for X on the horizontal axis would be 0. See Table A.12 for a full list of child properties.	*object*
right	The right face of the shape within the bounding box; the face at which the value for X on the horizontal axis would be N, where N equals the dimensional property which is on the X axis. See Table A.12 for a full list of child properties.	*object*

Required Properties: ***left*** and ***right***

Table A.11: The available properties for the complex geometrical faces object within a [regular] *element* in the PBSHM schema.

Property	Description	Type
dimensions	Each measurable dimension associated with the face being described. There is one child property for each given dimension. Child properties are either a named property (See Table A.14), or they are wildcard properties which can have any valid JSON string as their name. The value for the property is an *object* for which the child properties can be found in Table A.38. For a wildcard property, the accepted values are 'Wildcard' under 'Value Object Type'. For named properties, the accepted value type are declared in Table A.14.	*object*
translational	Y and Z translational values within the bounding box coordinate space to convey the translational movements of the face. See Table A.13 for a full list of child properties.	*object*

Required Properties: ***dimensions*** and ***translational***

Table A.12: The available properties for the complex geometrical face side object within a [regular] *element* in the PBSHM schema.

Property	Description	Type
y	The Y translational value within the bounding box coordinate space. See Table A.36 for the available child properties and, within the table, 'Linear' under 'Value Object Type' for the accepted values.	*object*
z	The Z translational value within the bounding box coordinate space. See Table A.36 for the available child properties and, within the table, 'Linear' under 'Value Object Type' for the accepted values.	*object*

Required Properties: ***y*** and ***z***

Additional Information: The values for the translational movements must be relative to the bounding box dimensions.

Table A.13: The available properties for the complex geometrical face translation object within a [regular] *element* in the PBSHM schema.

Type Tree	Dimension Property	Dimension Object
solid* → *translateAndScale* → *cuboid	***length***	*Linear Dimension*
solid* → *translateAndScale* → *cylinder	***length***	*Linear Dimension*
solid* → *translateAndScale* → *other	***length***	*Linear Dimension*
shell* → *translateAndScale* → *cuboid	***length***	*Linear Dimension*
shell* → *translateAndScale* → *cylinder	***length***	*Linear Dimension*
shell* → *translateAndScale* → *other	***length***	*Linear Dimension*

Type Tree	Face Dim. Property	Face Dim. Object
solid* → *translateAndScale* → *cuboid	***width***, ***height***	*Linear Dimension*
solid* → *translateAndScale* → *cylinder	***radius***	*Linear Dimension*
solid* → *translateAndScale* → *other		
shell* → *translateAndScale* → *cuboid	***thickness***, ***width***, ***height***	*Linear Dimension*
shell* → *translateAndScale* → *cylinder	***thickness***, ***radius***	*Linear Dimension*
shell* → *translateAndScale* → *other	***thickness***	*Linear Dimension*

Required Properties: Any named property in either the dimension properties or the face dimension properties in the Table above. See Table A.38 for the available child properties and, within the table, 'Linear' under 'Dimension Object Type' for the accepted values.

Additional Information: Additional dimensions can be declared as a *Wildcard*. See Table A.38 for the available child properties and, within the table, 'Linear' under 'Dimension Object Type' for the accepted values.

Table A.14: The available complex geometrical types and their associated required dimension properties, face dimension properties and accepted object types for a [regular] *element* in the PBSHM schema.

Material

Property	Description	Type
type	The classification of the material that composes the [regular] *element*. See figure A.1 for the list of accepted values.	*object*
symmetry	The symmetry of the physical properties of the material. Accepted values: 'isotropic'.	*string*
properties	Each measurable property of the material being described. See Table A.16	*array*

Required Properties: ***type***, ***symmetry*** if ***properties*** provided

Table A.15: The available properties for the material object within a [regular] *element* in the PBSHM schema.

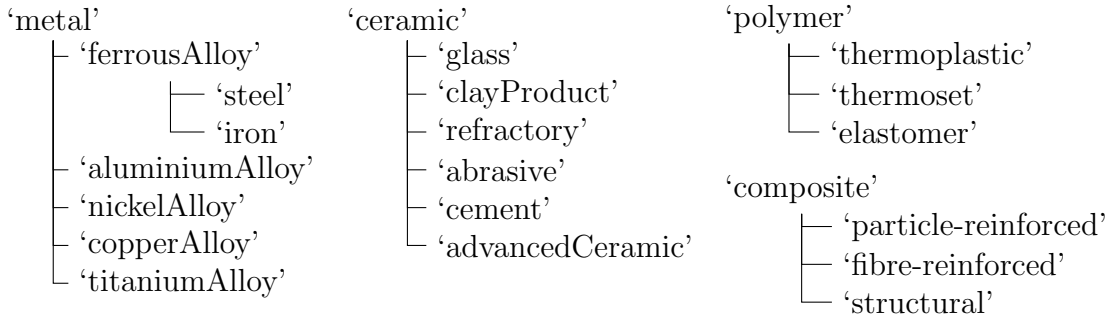


Figure A.1: The available material type tree for a [regular] *element* in the PBSHM schema.

Property	Description	Type
type	The type of property on the material within the [regular] *element*. See Tables A.17 and A.21 for accepted values which don't support a ***unit*** value and see Table A.18 for accepted values which support and require a ***unit*** value.	*string*
unit	The unit associated with the given value, if the ***type*** selected supports a ***unit*** property.	*string*
value	The given value for the material property. This is either a singular value (if supported) or a conditional value based upon test conditions provided. See Table A.19 for a list of accepted child properties for a conditional value. Singular values are supported by types in Tables A.17 and A.18 and conditional values are supported by types in Tables A.17, A.18 and A.21.	*int*, *double* or *object*

Required Properties: ***type***, ***value*** and ***unit*** if the associated ***type*** value is a unit based property as defined within Table A.18

Table A.16: The available properties for the material properties object within a [regular] *element* in the PBSHM schema.

Type
'poissonsRatio'
'elongation'
'reductionInArea'
'fatigueStrengthExponent'
'fatigueDuctilityCoefficient'
'fatigueDuctilityExponent'

Table A.17: The available standard unit-free material properties for a [regular] *element* in the PBSHM schema. The types in this Table, support both singular and conditional values.

Type	Unit
'density'	'kg/m^3', 'g/cm^3', 'kg/L', 'g/mL', 't/m^3', 'kg/dm^3', 'oz/cu in', 'other'
'linearThermalExpansionCoefficient'	'K^-1', 'C^-1', 'F^-1', 'other'
'volumetricThermalExpansionCoefficient'	'K^-1', 'C^-1', 'F^-1', 'other'
'youngsModulus'	'GPa', 'MPa', 'kPa', 'Pa', 'Mpsi', 'ksi', 'psi', 'other'
'shearModulus'	'GPa', 'MPa', 'kPa', 'Pa', 'Mpsi', 'ksi', 'psi', 'other'
'compressiveStrength'	'GPa', 'MPa', 'kPa', 'Pa', 'Mpsi', 'ksi', 'psi', 'other'
'shearStrength'	'GPa', 'MPa', 'kPa', 'Pa', 'Mpsi', 'ksi', 'psi', 'other'
'ultimateTensileStrength'	'GPa', 'MPa', 'kPa', 'Pa', 'Mpsi', 'ksi', 'psi', 'other'
'yieldStrength'	'GPa', 'MPa', 'kPa', 'Pa', 'Mpsi', 'ksi', 'psi', 'other'
'0.1%ProofStress'	'GPa', 'MPa', 'kPa', 'Pa', 'Mpsi', 'ksi', 'psi', 'other'
'fatigueStrengthCoefficient'	'GPa', 'MPa', 'kPa', 'Pa', 'Mpsi', 'ksi', 'psi', 'other'
'tensileToughness'	'GJ/m^3', 'MJ/m^3', 'kJ/m^3', 'J/m^3', 'ibf/in^3', 'other'
'fractureToughness'	'TPa/m^(1/2)', 'GPa/m^(1/2)', 'MPa/m^(1/2)', 'kPa/m^(1/2)', 'Pa/m^(1/2)', 'psi/in^(1/2)', 'other'

Table A.18: The available standard unit-based material properties and their associated accepted units for a [regular] *element* in the PBSHM schema. The types in this Table, support both singular and conditional values.

Property	Description	Type
environmental	The test environmental conditions associated with the given value. See Table A.20 for a full list of child properties.	*object*
parameters	The test parameters associated with the given value. There is one child property for each given test parameter associated with the given material value. Child properties are either a named property as declared as part of a complex material property type (See Table A.21), or they are wildcard properties which can have any valid JSON string as their name. The value for the property is an *object* for which the child properties can be found in Table A.36. For a wildcard property, the accepted values are 'Wildcard' under 'Value Object Type'. For named properties, the accepted value type is declared in Table A.21 which the accepted values are under 'Value Object Type' in Table A.36.	*object*
value	The singular value for the property at the given test ***environmental*** and ***parameters*** values.	*int* or *double*

Required Properties: ***value***, either ***environmental*** or ***parameters***

Table A.19: The available properties for the conditional material properties object for a [regular] *element* in the PBSHM schema.

Property	Description	Type
temperature	The temperature at which the test was conducted. See Table A.36 for the available child properties and, within the table, 'Temperature' under 'Value Object Type' for the accepted values.	*object*
humidity	The humidity at which the test was conducted. See Table A.36 for the available child properties and, within the table, 'Percentage' under 'Value Object Type' for the accepted values.	*object*

Required Properties: none

Additional Information: Additional environmental properties can be declared as a *Wildcard*. See Table A.36 for the available child properties and, within the table, 'Wildcard' under 'Value Object Type' for the accepted values..

Table A.20: The available properties for the environmental conditional material properties object for a [regular] *element* in the PBSHM schema.

Type	Named Property	Accepted Value Object Type
'vickersHardness'	***load***	*Force*
	duration	*Duration*
'brinellHardness'	***diameter***	*Brinell hardness diameter*
	ball	*Brinell hardness ball*
	force	*Force*

Table A.21: The available complex material properties and their associated required environmental properties and accepted values for a [regular] *element* in the PBSHM schema. The types in this Table, support only conditional values.

A.2.2 Ground element

Property	Description	Type
name	The unique name for the *element* within the *model*. Must have a length between 1 and 64 characters.	*string*
description	Additional non-structured information to describe the *element*.	*string*
type	The selected type of *element*. Accepted values: 'ground'.	*string*

Required Properties: ***name*** and ***type***

Table A.22: The available properties for the declaration of a [ground] *element* within the PBSHM schema.

A.2.3 Perfect & boundary relationship

Property	Description	Type
name	The unique name for the *relationship* within the *model*. Must have a length between 1 and 64 characters.	*string*
description	Additional non-structured information to describe the *relationship*.	*string*
type	Selected type of *relationship*. Accepted values: 'perfect' or 'boundary'.	*string*
elements	The list of *element*s which are part of this *relationship*. Must have exactly two *element*s in the list. Each member of the list is an *object*. See Table A.24 for the available child properties. If the *type* value is 'boundary', one of the *element*s must be a [ground] *element*.	*array*
coordinates	Position of the *relationship* within the *model*s coordinate space. See Table A.25 for the available child properties.	*object*

Required Properties: ***name***, ***type*** and ***elements***

Table A.23: The available properties for the declaration of a [perfect] and [boundary] *relationship* within the PBSHM schema.

Property	Description	Type
name	The name of the *element* within the current *model* which is part of this *relationship*. Must have a length between 1 and 64 characters.	*string*

Required Properties: ***name***

Table A.24: The available properties for named element object within a [perfect] and [boundary] *relationship* in the PBSHM schema.

Property	Description	Type
global	Translational coordinates within the global coordinate space. See Table A.26 for the available child properties.	*object*

Required Properties: ***global***

Table A.25: The available properties for the coordinate object within a [perfect] and [boundary] *relationship* in the PBSHM schema.

Property	Description	Type
translational	X, Y and Z translational values within the global coordinate space. See 'Translational coordinates Object' in Table A.35 for the available child properties.	*object*

Required Properties: ***translational***

Table A.26: The available properties for the global coordinate object within a [perfect] and [boundary] *relationship* in the PBSHM schema.

A.2.4 Connection relationship

Property	Description	Type
name	The unique name for the *relationship* within the *model*. Must have a length between 1 and 64 characters.	*string*
description	Additional none structured information to describe the *relationship*.	*string*
type	Selected type of *relationship*. Accepted values: 'connection'.	*string*
elements	The list of *element*s which are part of this *relationship* including the position of where the *relationship* resides on the *element*. Must have exactly two or more *element*s in the list. Each member of the list is an *object*. See Table A.28 for the available child properties.	*array*

Required Properties: ***name***, ***type*** and ***elements***

Table A.27: The available properties for the declaration of a [connection] *relationship* within the PBSHM schema.

Property	Description	Type
name	The name of the *element* within the current *model* which is part of this *relationship*. Must have a length between 1 and 64 characters.	*string*
nature	The nature of physics that would construct the joint. See Table A.30 for a list of child properties and see the values under the 'static' root of the tree in Figure A.2 for the branch of accepted values.	*object*
coordinates	Position of the *relationship*s interaction with the *element* within the *model*s coordinate space. See Table A.25 for the available child properties.	*object*

Required Properties: ***name*** and ***nature***

Table A.28: The available properties for named element object within a [connection] *relationship* in the PBSHM schema.

A.2.5 Joint relationship

Property	Description	Type
name	The unique name for the *relationship* within the *model*. Must have a length between 1 and 64 characters.	*string*
description	Additional non-structured information to describe the *relationship*.	*string*
type	Selected type of *relationship*. Accepted values: 'joint'.	*string*
nature	The nature of physics that construct the joint. See Table A.30 for a list of child properties and see Figure A.2 for a tree of accepted values.	*object*
degreesOfFreedom	The allowed movement within the joint. See Table A.31 for a list of child properties. Only available when the root value of the ***nature*** tree is 'dynamic'.	*object*
elements	The list of *element*s which are part of this *relationship*. Must have exactly two *element*s in the list. Each member of the list is an *object*. See Table A.33 for the available child properties.	*array*

Required Properties: ***name***, ***type***, ***nature*** and ***elements***

Table A.29: The available properties for the declaration of a [joint] *relationship* within the PBSHM schema.

Property	Description	Type
name	The nature of the [joint] or [connection] *relationship*.	*string*
nature	The child nature of the current tree level. See Table A.30 for the accepted child properties.	*string*

Required Properties: ***name***

Table A.30: The available properties for a nested nature object within a [joint] or [connection] *relationship* in the PBSHM schema.

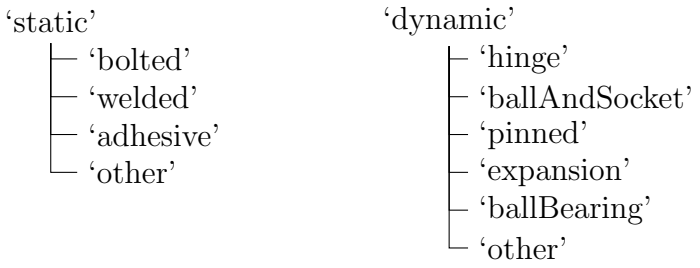


Figure A.2: The available nature type tree within a [joint] or [connection] *relationship* in the PBSHM schema. When declaring the nature within a [connection] *relationship*, only the 'static' portion of the nature tree is available.

Property	Description	Type
global	Translational and Rotational degrees of freedom of the joint. See Table A.32 for the available child properties.	*object*

Required Properties: ***global***

Table A.31: The available properties for the degrees of freedom object within a [dynamic] [joint] *relationship* in the PBSHM schema.

Property	Description	Type
translational	X, Y and Z translational degrees of freedom values within the global coordinate space. See 'Bounded Translational coordinates Object' in Table A.35 for the available child properties.	*object*
rotational	Alpha, Beta and Gamma rotational degrees of freedom values within the global coordinate space. See 'Bounded Rotational Coordinates Object' in Table A.35 for the available child properties.	*object*

Required Properties: ***translational*** or ***rotational***

Table A.32: The available properties for the global degrees of freedom object within a [dynamic] [joint] *relationship* in the PBSHM schema.

Property	Description	Type
name	The name of the *element* within the current *model* which is part of this *relationship*. Must have a length between 1 and 64 characters.	*string*
coordinates	Position of the *relationship*'s interaction with the *element* within the *model*s coordinate space. See Table A.25 for the available child properties.	*object*

Required Properties: ***name***

Table A.33: The available properties for named element object within a [joint] *relationship* in the PBSHM schema.

A.2.6 Shared objects

Property	Description	Type
name	The name of the type at the current tree level.	*string*
type	The child type of current tree level. See Table A.34 for the accepted child properties.	*object*

Required Properties: ***name***

Table A.34: The available properties for a nested type object for embedding type trees within the PBSHM schema.

Translational Coordinates Object

Property	Description	Type
x, ***y***, ***z***	The translational values within the coordinate space. See Table A.36 for the available child properties and, within the table, 'Linear' under 'Value Object Type' for the accepted values.	*object*

Required Properties: ***x***, ***y*** and ***z***

Rotational Coordinates Object

Property	Description	Type
alpha, ***beta***, ***gamma***	The rotational values within the coordinate space. See Table A.36 for the available child properties and, within the table, 'Angular' under 'Value Object Type' for the accepted values.	*object*

Required Properties: ***alpha***, ***beta*** and ***gamma***

Bounded Translational Coordinates Object

Property	Description	Type
x, ***y***, ***z***	The translational values within the coordinate space. See Table A.37 for the available child properties and, within the table, 'Linear' under 'Bounded Value Object Type' for the accepted values.	*object*

Required Properties: ***x***, ***y*** and ***z***

Bounded Rotational Coordinates Object

Property	Description	Type
alpha, ***beta***, ***gamma***	The rotational values within the coordinate space. See Table A.37 for the available child properties and, within the table, 'Angular' under 'Bounded Value Object Type' for the accepted values.	*object*

Required Properties: ***alpha***, ***beta*** and ***gamma***

Table A.35: The available properties for a coordinate object and associated accepted values for different types of coordinate object within the PBSHM schema.

Property	Description	Type
unit	The unit associated with the given ***value***. See below for the list of accepted values given the type of value object.	*string*
value	The value for the measurement.	*int* or *double*

Required Properties: ***unit*** and ***value***

Value Object Type	Accepted Unit Values
Linear	'mm', 'cm', 'm', 'km' or 'other'
Angular	'degrees', 'radians' or 'other'
Temperature	'C', 'F', 'K' or 'other'
Force	'kgf'
Duration	's'
Percentage	'%'
Wildcard	Any valid string
Brinell hardness diameter	'mm'
Brinell hardness ball	'W' or 'S'

Table A.36: The available properties for a value object and associated accepted values for the different types of value object within the PBSHM schema.

Property	Description	Type
unit	The unit associated with the given ***minimum*** and ***maximum*** values. See below for the list of accepted values given the type of value object.	*string*
minimum	The minimum value for the degree of freedom	*int* or *double*
maximum	The maximum value for the degree of freedom	*int* or *double*

Required Properties: ***unit***, ***minimum*** and ***maximum***

Bounded Value Object Type	Accepted Unit Values
Linear	'mm', 'cm', 'm', 'km' or 'other'
Angular	'degrees', 'radians' or 'other'

Table A.37: The available properties for a bounded value object and associated accepted values for the different types of bounded value object within the PBSHM schema.

Property	Description	Type
axis	The axis in which the dimension has been measured. Accepted values: 'x', 'y', 'z', 'xy', 'xz' or 'yz'	*string*
source	The source of which the measurement has come from. Accepted values: 'measured' or 'nominal'	*string*
unit	The unit associated with the given measurement ***value***. See below for the list of accepted values given the type of value object.	*string*
value	The value for the measurement.	*int* or *double*

Required Properties: ***axis***, ***source***, ***unit*** and ***value***

Dimension Object Type	Accepted Unit Values
Linear	'mm', 'cm', 'm', 'km' or 'other'
Angular	'degrees', 'radians' or 'other'
Wildcard	Any valid string

Table A.38: The available properties for a dimension object and associated accepted values for the different types of dimension object within the PBSHM schema.

A.3 Channel Schema

Property	Description	Type
name	Name of the channel, must be unique within the structure.	*string*
type	The selected type value for this channel, see Table. A.40 for the list of available options.	*string*
unit	The selected unit value of this channel on which the value is based; see Table. A.40 for the list of available options for the selected type.	*string*
value	Value of the channel, stored in the selected unit.	*See Table. A.40*

Required Properties: ***name***, ***type***, ***value*** and ***unit*** dependent upon the selected ***type***

Table A.39: List of channel object properties in the PBSHM Schema.

Types	Units	Values
acceleration	*m/s^2*, *g*, *v*, *other*	*int*, *double*, *object*
velocity	*m/s*, *v*, *other*	*int*, *double*, *object*
displacement	*mm*, *cm*, *m*, *km*, *other*	*int*, *double*, *object*
angularAcceleration	*degrees/s^2*, *radians/s^2*, *other*	*int*, *double*, *object*
angularVelocity	*degrees/s*, *radians/s*, *other*	*int*, *double*, *object*
angularDisplacement	*degrees*, *radians*, *other*	*int*, *double*, *object*
tilt	*degrees*, *radians*, *other*	*int*, *double*, *object*
strain	*nd*, *other*	*int*, *double*, *object*
tension	*fN*, *pN*, *nN*, *μN*, *mN*, *cN*, *dN*, *N*, *daN*, *hN*, *kN*, *MN*, *GN*, *TN*, *PN*, *other*	*int*, *double*, *object*
load	*fN*, *pN*, *nN*, *μN*, *mN*, *cN*, *dN*, *N*, *daN*, *hN*, *kN*, *MN*, *GN*, *TN*, *PN*, *other*	*int*, *double*, *object*
structuralPotentialHydrogen	*pH*, *other*	*int*, *double*, *object*
temperature	*C*, *F*, *K*, *other*	*int*, *double*, *object*
humidity	*%*, *other*	*int*, *double*, *object*
speed	*mph*, *ft/s*, *km/h*, *m/s*, *kn*, *other*	*int*, *double*, *object*
direction	*degrees*, *radians*, *other*	*int*, *double*, *object*

pressure	*fPa*, *pPa*, *nPa*, *μPa*, *mPa*, *cPa*, *dPa*, *Pa*, *daPa*, *hPa*, *kPa*, *MPa*, *GPa*, *TPa*, *PPa*, *at*, *atm*, *bar*, *psi*, *other*	*int*, *double*, *object*
altitude	*mm*, *cm*, *m*, *km*, *feet*, *other*	*int*, *double*, *object*
pitch	*degrees*, *radians*, *other*	*int*, *double*, *object*
yaw	*degrees*, *radians*, *other*	*int*, *double*, *object*
roll	*degrees*, *radians*, *other*	*int*, *double*, *object*
pitchRate	*degrees/s*, *radians/s*, *other*	*int*, *double*, *object*
yawRate	*degrees/s*, *radians/s*, *other*	*int*, *double*, *object*
rollRate	*degrees/s*, *radians/s*, *other*	*int*, *double*, *object*
current	*fA*, *pA*, *nA*, *μA*, *mA*, *cA*, *dA*, *A*, *daA*, *hA*, *kA*, *MA*, *GA*, *TA*, *PA*, *other*	*int*, *double*, *object*
charge	*fC*, *pC*, *nC*, *μC*, *mC*, *cC*, *dC*, *C*, *daC*, *hC*, *kC*, *MC*, *GC*, *TC*, *PC*, *other*	*int*, *double*, *object*
power	*fW*, *pW*, *nW*, *μW*, *mW*, *cW*, *dW*, *W*, *daW*, *hW*, *kW*, *MW*, *GW*, *TW*, *PW*, *other*	*int*, *double*, *object*
voltage	*fV*, *pV*, *nV*, *μV*, *mV*, *cV*, *dV*, *V*, *daV*, *hV*, *kV*, *MV*, *GV*, *TV*, *PV*, *other*	*int*, *double*, *object*
resistance	*fΩ*, *pΩ*, *nΩ*, *μΩ*, *mΩ*, *cΩ*, *dΩ*, *Ω*, *daΩ*, *hΩ*, *kΩ*, *MΩ*, *GΩ*, *TΩ*, *PΩ*, *other*	*int*, *double*, *object*
capacitance	*fF*, *pF*, *nF*, *μF*, *mF*, *cF*, *dF*, *F*, *daF*, *hF*, *kF*, *MF*, *GF*, *TF*, *PF*, *other*	*int*, *double*, *object*

inductance	*fH*, *pH*, *nH*, *μH*, *mH*, *cH*, *dH*, *H*, *daH*, *hH*, *kH*, *MH*, *GH*, *TH*, *PH*, *other*	*int*, *double*, *object*
frequency	*fHz*, *pHz*, *nHz*, *μHz*, *mHz*, *cHz*, *dHz*, *Hz*, *daHz*, *hHz*, *kHz*, *MHz*, *GHz*, *THz*, *PHz*, *other*	*int*, *double*, *object*
conductance	*fS*, *pS*, *nS*, *μS*, *mS*, *cS*, *dS*, *S*, *daS*, *hS*, *kS*, *MS*, *GS*, *TS*, *PS*, *other*	*int*, *double*, *object*
magneticFlux	*fWb*, *pWb*, *nWb*, *μWb*, *mWb*, *cWb*, *dWb*, *Wb*, *daWb*, *hWb*, *kWb*, *MWb*, *GWb*, *TWb*, *PWb*, *other*	*int*, *double*, *object*
magneticFieldStrength	*fT*, *pT*, *nT*, *μT*, *mT*, *cT*, *dT*, *T*, *daT*, *hT*, *kT*, *MT*, *GT*, *TT*, *PT*, *other*	*int*, *double*, *object*
integer	n/a	*int*
double	n/a	*double*
text	n/a	*string*
date	n/a	*long*

Table A.40: The available values for Channel properties within the PBSHM schema.

Property	Description	Type
min	Minimum channel value over the observed time period.	*int*, *double*
max	Maximum channel value over the observed time period.	*int*, *double*
mean	Mean channel value over the observed time period.	*int*, *double*
std	Standard deviation channel value over the observed time period.	*int*, *double*

Required Properties: At least two of the properties described above.

Table A.41: List of value object properties in the PBSHM Schema.

Appendix B

Real-world aircraft

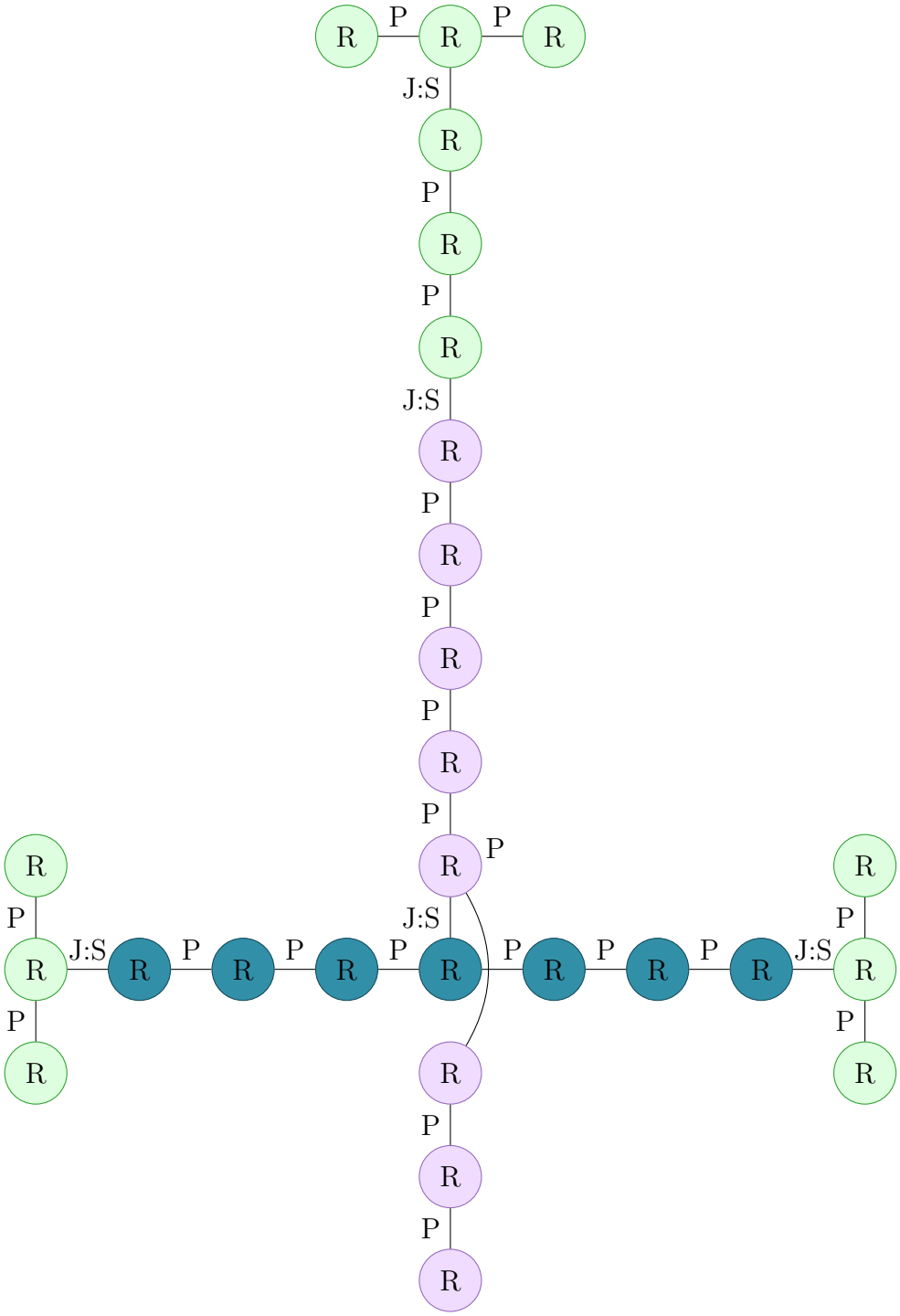


Figure B.1: The AG representation of the GARTEUR IE model described within this thesis without using the proposed PBSHM AG notations.

Vertical Stabiliser

Element Names	Type Attributes
vertical-stabiliser-a, vertical-stabiliser-b, vertical-stabiliser-c, vertical-stabiliser-d	Contextual: 'aerofoil' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'
rudder	Contextual: 'other' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'

Left Wing

Element Names	Type Attributes
left-wing-a, left-wing-b, left-wing-c, left-wing-d, left-wing-e, left-wing-f, left-wing-g	Contextual: 'wing' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'
left-flap-a, left-flap-b, left-flap-c, left-flap-d, left-flap-e	Contextual: 'aerofoil' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'
left-aileron	Contextual: 'aerofoil' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'

Left Horizontal Stabiliser

Element Names	Type Attributes
left-horizontal-stabiliser-a, left-horizontal-stabiliser-b	Contextual: 'aerofoil' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'

Right Horizontal Stabiliser

Element Names	Type Attributes
right-horizontal-stabiliser-a, right-horizontal-stabiliser-b	Contextual: 'aerofoil' Material: 'metal' →'aluminiumAlloy' Geometry: 'shell' →'translateAndScale' →'cylinder'

Table B.1: [regular] *element*s for the left wing, vertical stabiliser, and horizontal stabilisers of the generalised Hawk T.Mk1.

Right Landing Gear Element Names	Type Attributes
right-shock-absorber	Contextual: ‘other’ Material: ‘metal’ →‘aluminiumAlloy’ Geometry: ‘solid’ →‘translateAndScale’ →‘cylinder’
right-support	Contextual: ‘other’ Material: ‘metal’ →‘aluminiumAlloy’ Geometry: ‘solid’ →‘translateAndScale’ →‘cylinder’
right-wheel	Contextual: ‘wheel’ Material: ‘other’ Geometry: ‘other’

Nose Landing Gear Element Names	Type Attributes
center-shock-absorber	Contextual: ‘other’ Material: ‘metal’ →‘aluminiumAlloy’ Geometry: ‘solid’ →‘translateAndScale’ →‘cylinder’
center-support	Contextual: ‘other’ Material: ‘metal’ →‘aluminiumAlloy’ Geometry: ‘solid’ →‘translateAndScale’ →‘cylinder’
center-wheel	Contextual: ‘wheel’ Material: ‘other’ Geometry: ‘other’

Left Landing Gear Element Names	Type Attributes
left-shock-absorber	Contextual: ‘other’ Material: ‘metal’ →‘aluminiumAlloy’ Geometry: ‘solid’ →‘translateAndScale’ →‘cylinder’
left-support	Contextual: ‘other’ Material: ‘metal’ →‘aluminiumAlloy’ Geometry: ‘solid’ →‘translateAndScale’ →‘cylinder’
left-wheel	Contextual: ‘wheel’ Material: ‘other’ Geometry: ‘other’

Table B.2: [regular] *element*s for the landing gear of the generalised Hawk T.Mk1.

Element Name
right-ground
left-ground
center-ground

Table B.3: [ground] *element*s for the landing gear of the generalised Hawk T.Mk1.

Left Wing

Relationship Name	Element Set	Type
left-wing-a-b	{left-wing-a, left-wing-b}	[perfect]
left-wing-b-c	{left-wing-b, left-wing-c}	[perfect]
left-wing-c-d	{left-wing-c, left-wing-d}	[perfect]
left-wing-d-e	{left-wing-d, left-wing-e}	[perfect]
left-wing-e-f	{left-wing-e, left-wing-f}	[perfect]
left-wing-f-g	{left-wing-f, left-wing-g}	[perfect]
left-wing-flap-a	{left-wing-a, left-flap-a}	[joint], [dynamic]
left-wing-flap-b	{left-wing-b, left-flap-b}	[joint], [dynamic]
left-wing-flap-c	{left-wing-c, left-flap-c}	[joint], [dynamic]
left-wing-flap-d	{left-wing-d, left-flap-d}	[joint], [dynamic]
left-wing-e-aileron	{left-wing-e, left-aileron}	[joint], [dynamic]
left-wing-f-aileron	{left-wing-f, left-aileron}	[joint], [dynamic]
left-wing-g-aileron	{left-wing-g, left-aileron}	[joint], [dynamic]
left-wing-a-flap-e	{left-wing-a, left-flap-e}	[joint], [dynamic]
left-wing-b-flap-e	{left-wing-b, left-flap-e}	[joint], [dynamic]
left-wing-c-flap-e	{left-wing-c, left-flap-e}	[joint], [dynamic]
left-wing-d-flap-e	{left-wing-d, left-flap-e}	[joint], [dynamic]
left-wing-a-fuselage-f	{left-wing-a, fuselage-f}	[joint], [static]

Vertical Stabiliser

Relationship Name	Element Set	Type
vertical-stabiliser-a-b	{vertical-stabiliser-a, vertical-stabiliser-b}	[perfect]
vertical-stabiliser-b-c	{vertical-stabiliser-b, vertical-stabiliser-c}	[perfect]
vertical-stabiliser-b-d	{vertical-stabiliser-b, vertical-stabiliser-d}	[joint], [static]
vertical-stabiliser-b-rudder	{vertical-stabiliser-b, rudder}	[joint], [dynamic]

Table B.4: *relationship*s for the left wing and vertical stabiliser of the generalised Hawk T.Mk1.

Fuselage to Vertical Stabiliser

Relationship Name	Element Set	Type
fuselage-h-vertical-stabiliser-a	{fuselage-h, vertical-stabiliser-a}	[joint], [static]
fuselage-i-vertical-stabiliser-b	{fuselage-i, vertical-stabiliser-b}	[joint], [static]
fuselage-j-vertical-stabiliser-b	{fuselage-j, vertical-stabiliser-b}	[joint], [static]
fuselage-j-vertical-stabiliser-d	{fuselage-j, vertical-stabiliser-d}	[joint], [static]
fuselage-k-vertical-stabiliser-d	{fuselage-k, vertical-stabiliser-d}	[joint], [static]
fuselage-l-vertical-stabiliser-d	{fuselage-k, vertical-stabiliser-d}	[joint], [static]

Right Horizontal Stabiliser to Fuselage

Relationship Name	Element Set & Type
right-horizontal-stabiliser-a-b	Set: {right-horizontal-stabiliser-a, right-horizontal-stabiliser-b} Type: [perfect]
right-horizontal-stabiliser-a-fuselage-k	Set: {right-horizontal-stabiliser-a, fuselage-k} Type: [joint], [dynamic]

Left Horizontal Stabiliser to Fuselage

Relationship Name	Element Set & Type
left-horizontal-stabiliser-a-b	Set: {left-horizontal-stabiliser-a, left-horizontal-stabiliser-b} Type: [perfect]
left-horizontal-stabiliser-a-fuselage-k	Set: {left-horizontal-stabiliser-a, fuselage-k} Type: [joint], [dynamic]

Table B.5: *relationship*s for the fuselage to vertical stabiliser and fuselage to horizontal stabilisers of the generalised Hawk T.Mk1.

Right Landing Gear Relationship Name	Element Set	Type
right-shock-absorber-wing-b	{right-shock-absorber, right-wing-b}	[joint], [dynamic]
right-shock-absorber-support	{right-shock-absorber, right-support}	[joint], [dynamic]
right-support-wheel	{right-support, right-wheel}	[joint], [dynamic]
right-wheel-ground	{right-wheel, right-ground}	[boundary]

Nose Landing Gear Relationship Name	Element Set	Type
center-shock-absorber-fuselage-b	{center-shock-absorber, fuselage-b}	[joint], [dynamic]
center-shock-absorber-support	{center-shock-absorber, center-support}	[joint], [dynamic]
center-support-wheel	{center-support, center-wheel}	[joint], [dynamic]
center-wheel-ground	{center-wheel, center-ground}	[boundary]

Left Landing Gear Relationship Name	Element Set	Type
left-shock-absorber-wing-b	{left-shock-absorber, left-wing-b}	[joint], [dynamic]
left-shock-absorber-support	{left-shock-absorber, left-support}	[joint], [dynamic]
left-support-wheel	{left-support, left-wheel}	[joint], [dynamic]
left-wheel-ground	{left-wheel, left-ground}	[boundary]

Table B.6: [relationship]s for the right, centre and left and landing gear of the generalised Hawk T.Mk1.

Bibliography

[1] C. R Farrar and K Worden. *Structural Health Monitoring: A Machine Learning Perspective*. Wiley, Chichester, West Sussex, U.K. ; Hoboken, N.J, 2013. ISBN 978-1-119-99433-6.

[2] C. R Farrar and K Worden. An introduction to structural health monitoring. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1851):303–315, 2007. ISSN 1364-503X, 1471-2962. doi: 10.1098/rsta.2006.1928.

[3] A Rytter. *Vibration Based Inspection of Civil Engineering Structures Ph. D.* PhD thesis, 1993.

[4] K Worden and J. M Dulieu-Barton. An Overview of Intelligent Fault Detection in Systems and Structures. *Structural Health Monitoring*, 3(1):85–98, 2004. ISSN 1475-9217, 1741-3168. doi: 10.1177/1475921704041866.

[5] L Bull, P Gardner, J Gosliga, T Rogers, N Dervilis, E Cross, E Papatheou, A Maguire, C Campos, and K Worden. Foundations of Population-based SHM, Part I: Homogeneous populations and forms. *Mechanical Systems and Signal Processing*, 148:107141, 2021. ISSN 08883270. doi: 10.1016/j.ymssp.2020.107141.

[6] J Gosliga, P Gardner, L Bull, N Dervilis, and K Worden. Foundations of Population-based SHM, Part II: Heterogeneous populations – Graphs, networks, and communities. *Mechanical Systems and Signal Processing*, 148: 107144, 2021. ISSN 08883270. doi: 10.1016/j.ymssp.2020.107144.

[7] P Gardner, L Bull, J Gosliga, N Dervilis, and K Worden. Foundations of Population-based SHM, Part III: Heterogeneous populations – Mapping and transfer. *Mechanical Systems and Signal Processing*, 149:107142, 2021. ISSN 08883270. doi: 10.1016/j.ymssp.2020.107142.

[8] G Tsialiamanis, C Mylonas, E Chatzi, N Dervilis, D. J Wagg, and K Worden. Foundations of Population-based SHM, Part IV: Structures and Feature Spaces as Geometry. page 57, 2021.

[9] S. J Pan and Q Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. ISSN 1041-4347. doi: 10.1109/TKDE.2009.191.

[10] K Weiss, T. M Khoshgoftaar, and D Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016. ISSN 2196-1115. doi: 10.1186/s40537-016-0043-6.

[11] A. L Barabási and M Pósfai. *Network Science*. Cambridge University Press, Cambridge, United Kingdom, 2016. ISBN 978-1-107-07626-6.

[12] M. E. J Newman. *Networks*. Oxford University Press, Oxford, second edition, 2018. ISBN 978-0-19-880509-0. doi: 10.1093/oso/9780198805090.001.0001.

[13] E Papatheou, N Dervilis, A. E Maguire, I Antoniadou, and K Worden. A Performance Monitoring Approach for the Novel Lillgrund Offshore Wind Farm. *IEEE Transactions on Industrial Electronics*, 62(10):6636–6644, 2015. ISSN 0278-0046, 1557-9948. doi: 10.1109/TIE.2015.2442212.

[14] I Antoniadou, N Dervilis, E Papatheou, A. E Maguire, and K Worden. Aspects of structural health and condition monitoring of offshore wind turbines. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373(2035):20140075, 2015. ISSN 1364-503X, 1471-2962. doi: 10.1098/rsta.2014.0075.

[15] K Worden, E. J Cross, N Dervilis, E Papatheou, and I Antoniadou. Structural Health Monitoring: From Structures to Systems-of-Systems. *IFAC-PapersOnLine*, 48(21):1–17, 2015. ISSN 24058963. doi: 10.1016/j.ifacol.2015.09.497.

[16] L. A Bull, T. J Rogers, N Dervilis, E. J Cross, and K Worden. A Gaussian Process Form for Population-Based Structural Health Monitoring.

[17] J Gosliga and K Worden. A General Representation for Assessing the Similarity of Structures. *Structural Health Monitoring 2019*, 2019.

[18] L. A Bull, P. A Gardner, J Gosliga, N Dervilis, E Papatheou, A. E Maguire, C Campos, T. J Rogers, E. J Cross, and K Worden. Towards Population-Based Structural Health Monitoring, Part I: Homogeneous Populations and Forms. In Z Mao, editor, *Model Validation and Uncertainty Quantification, Volume 3*, pages 287–302. Springer International Publishing, Cham, 2020. ISBN 978-3-030-48778-2 978-3-030-47638-0. doi: 10.1007/978-3-030-47638-0_32.

[19] J Gosliga, P Gardner, L. A Bull, N Dervilis, and K Worden. Towards Population-Based Structural Health Monitoring, Part II: Heterogeneous Populations and Structures as Graphs. In B Dilworth and M Mains, editors, *Topics in Modal Analysis & Testing, Volume 8*, pages 177–187. Springer International Publishing, Cham, 2021. ISBN 978-3-030-47716-5 978-3-030-47717-2. doi: 10.1007/978-3-030-47717-2_17.

[20] J Gosliga, P Gardner, L. A Bull, N Dervilis, and K Worden. Towards Population-Based Structural Health Monitoring, Part III: Graphs, Networks and Communities. In B Dilworth and M Mains, editors, *Topics in Modal Analysis & Testing, Volume 8*, pages 255–267. Springer International Publishing, Cham, 2021. ISBN 978-3-030-47716-5 978-3-030-47717-2. doi: 10.1007/978-3-030-47717-2_26.

[21] P Gardner, L. A Bull, J Gosliga, N Dervilis, and K Worden. Towards Population-Based Structural Health Monitoring, Part IV: Heterogeneous Populations, Transfer and Mapping. In Z Mao, editor, *Model Validation and Uncertainty Quantification, Volume 3*, pages 187–199. Springer International Publishing, Cham, 2020. ISBN 978-3-030-48778-2 978-3-030-47638-0. doi: 10.1007/978-3-030-47638-0_20.

[22] K Worden. Towards Population-Based Structural Health Monitoring, Part VI: Structures as Geometry. In S Pakzad, editor, *Dynamics of Civil Structures, Volume 2*, pages 221–236. Springer International Publishing, Cham, 2021. ISBN 978-3-030-47633-5 978-3-030-47634-2. doi: 10.1007/978-3-030-47634-2_26.

[23] W Lin, K Worden, A Eoghan Maguire, and E. J Cross. Towards Population-Based Structural Health Monitoring, Part VII: EOV Fields – Environmental

Mapping. In B Dilworth and M Mains, editors, *Topics in Modal Analysis & Testing, Volume 8*, pages 297–304. Springer International Publishing, Cham, 2021. ISBN 978-3-030-47716-5 978-3-030-47717-2. doi: 10.1007/978-3-030-47717-2_31.

[24] P Jaccard. Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bull Soc Vaudoise Sci Nat*, 37:241–272, 1901.

[25] S. E Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. ISSN 15740137. doi: 10.1016/j.cosrev.2007.05.001.

[26] E Duesbury, J. D Holliday, and P Willett. Maximum Common Subgraph Isomorphism Algorithms.

[27] S. N Ndiaye and C Solnon. CP Models for Maximum Common Subgraph Problems. In J Lee, editor, *Principles and Practice of Constraint Programming – CP 2011*, volume 6876, pages 637–644. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-23785-0 978-3-642-23786-7. doi: 10.1007/978-3-642-23786-7_48.

[28] C Bron and J Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973. ISSN 0001-0782, 1557-7317. doi: 10.1145/362342.362367.

[29] S Rogers and M Girolami. *A First Course in Machine Learning*. Chapman & Hall/CRC Machine Learning & Pattern Recognition Series. CRC Press, Taylor & Francis Group, Boca Raton London New York, second edition, first issued in paperback edition, 2020. ISBN 978-1-4987-3848-4 978-0-367-57464-2.

[30] K Worden and G Manson. The application of machine learning to structural health monitoring. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1851):515–537, 2007. ISSN 1364-503X, 1471-2962. doi: 10.1098/rsta.2006.1938.

[31] N Steenrod. *The Topology of Fibre Bundles*, volume 44. Princeton university press, 1999.

[32] K Worden, L. A Bull, P Gardner, J Gosliga, T. J Rogers, E. J Cross, E Papatheou, W Lin, and N Dervilis. A Brief Introduction to Recent Developments in Population-Based Structural Health Monitoring. *Frontiers*

in Built Environment, 6:146, 2020. ISSN 2297-3362. doi: 10.3389/fbuil.2020.00146.

[33] J Gosliga, A Bunce, D Hester, and K Worden. Creating a network of structures based on physical similarity. 2021.

[34] G. W Milligan and M. C Cooper. Methodology Review: Clustering Methods. *Applied Psychological Measurement*, 11(4):329–354, 1987. ISSN 0146-6216, 1552-3497. doi: 10.1177/014662168701100401.

[35] K Worden, D Hester, A Bunce, and J Gosliga. When is a bridge not an aeroplane? In Á Cunha and E Caetano, editors, *Proceedings of the 10th International Conference on Structural Health Monitoring of Intelligent Infrastructure (SHMII-10)*, International Conference on Structural Health Monitoring of Intelligent Infrastructure: Proceedings, pages 1775–1782. International Society for Structural Health Monitoring of Intelligent Infrastructure, ISHMII, 2021.

[36] A Bunce, D Hester, K Worden, J Gosliga, and S Taylor. PBSHM - guidance for ensuring quality when creating an IE model for a bridge. In Á Cunha and E Caetano, editors, *Proceedings of the 10th International Conference on Structural Health Monitoring of Intelligent Infrastructure (SHMII-10)*, International Conference on Structural Health Monitoring of Intelligent Infrastructure: Proceedings, pages 1789–1795. International Society for Structural Health Monitoring of Intelligent Infrastructure, ISHMII, 2021.

[37] D Hester, A Bunce, K Worden, J Gosliga, and S Taylor. Comparison of bridge topology before and after repair using attributed graph comparisons towards population based SHM. In Á Cunha and E Caetano, editors, *Proceedings of the 10th International Conference on Structural Health Monitoring of Intelligent Infrastructure (SHMII-10)*, International Conference on Structural Health Monitoring of Intelligent Infrastructure: Proceedings, pages 1797–1802. International Society for Structural Health Monitoring of Intelligent Infrastructure, ISHMII, 2021.

[38] J Gosliga, D Hester, K Worden, and A Bunce. On Population-based structural health monitoring for bridges. *Mechanical Systems and Signal Processing*, 173: 108919, 2022. ISSN 08883270. doi: 10.1016/j.ymssp.2022.108919.

[39] J. W Raymond and P Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. 2002.

[40] D Fooshee, A Andronico, and P Baldi. ReactionMap: An Efficient Atom-Mapping Algorithm for Chemical Reactions. *Journal of Chemical Information and Modeling*, 53(11):2812–2819, 2013. ISSN 1549-9596, 1549-960X. doi: 10.1021/ci400326p.

[41] Y Cao, T Jiang, and T Girke. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics*, 24(13):i366–i374, 2008. ISSN 1367-4811, 1367-4803. doi: 10.1093/bioinformatics/btn186.

[42] H.-C Ehrlich and M Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: A review. *WIREs Computational Molecular Science*, 1(1):68–79, 2011. ISSN 1759-0876, 1759-0884. doi: 10.1002/wcms.5.

[43] L Schietgat, J Ramon, and M Bruynooghe. A polynomial-time maximum common subgraph algorithm for outerplanar graphs and its application to chemoinformatics. *Annals of Mathematics and Artificial Intelligence*, 69(4):343–376, 2013. ISSN 1012-2443, 1573-7470. doi: 10.1007/s10472-013-9335-0.

[44] Y Park, D Reeves, V Mulukutla, and B Sundaravel. Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–4, Oak Ridge Tennessee USA, 2010. ACM. ISBN 978-1-4503-0017-9. doi: 10.1145/1852666.1852716.

[45] Y Park and D Reeves. Deriving common malware behavior through graph clustering. 2011.

[46] K Shearer, H Bunke, and S Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34(5):1075–1091, 2001. ISSN 00313203. doi: 10.1016/S0031-3203(00)00048-0.

[47] A Hati, S Chaudhuri, and R Velmurugan. Image co-segmentation using maximum common subgraph matching and region co-growing. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, the Netherlands, October 11-14, 2016, Proceedings, Part VI 14*, pages 736–752. Springer, 2016.

[48] H Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997. ISSN 01678655. doi: 10.1016/S0167-8655(97)00060-3.

[49] M.-L Fernández and G Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6-7):753–758, 2001. ISSN 01678655. doi: 10.1016/S0167-8655(01)00017-4.

[50] E. F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, page 11, 1970.

[51] C Batini, M Lenzerini, and S. B Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986. ISSN 0360-0300, 1557-7341. doi: 10.1145/27633.27634.

[52] M. L Brodie and D Ridjanovic. On the Design and Specification of Database Transactions. In *Readings in Artificial Intelligence and Databases*, pages 185–206. Elsevier, 1989. ISBN 978-0-934613-53-8. doi: 10.1016/B978-0-934613-53-8.50018-2.

[53] A Davoudian, L Chen, and M Liu. A Survey on NoSQL Stores. *ACM Computing Surveys*, 51(2):1–43, 2019. ISSN 0360-0300, 1557-7341. doi: 10.1145/3158661.

[54] C. J Date. A critique of the SQL database language. *ACM SIGMOD Record*, 14(3):8–54, 1984. ISSN 0163-5808. doi: 10.1145/984549.984551.

[55] J Funk. What's behind technological hype? *Issues in Science and Technology*, 36(1):36–42, 2019.

[56] C Nance, T Losser, R Iype, and G Harmon. NOSQL VS RDBMS - Why there is room for both. page 7, 2013.

[57] R Zafar, E Yafi, M. F Zuhairi, and H Dao. Big Data: The NoSQL and RDBMS review. In *2016 International Conference on Information and Communication Technology (ICICTM)*, pages 120–126, Kuala Lumpur, Malaysia, 2016. IEEE. ISBN 978-1-5090-0412-6. doi: 10.1109/ICICTM.2016.7890788.

[58] A Gandini, M Gribaudo, W. J Knottenbelt, R Osman, and P Piazzolla. Performance evaluation of NoSQL databases. page 15.

[59] S Bradshaw, E Brazil, and K Chodorow. *MongoDB: The Definitive Guide*. O'Reilly, third edition edition, 2019. ISBN 978-1-4919-5446-1.

[60] MongoDB, Inc. Encryption at Rest - MongoDB Manual. https://docs.mongodb.com/manual/core/security-encryption-at-rest/, 2020.

[61] MongoDB, Inc. Users - MongoDB Manual. https://docs.mongodb.com/manual/core/security-users/\#sharded-cluster-users, 2020.

[62] MongoDB, Inc. TLS/SSL (Transport Encryption) - MongoDB Manual. https://docs.mongodb.com/manual/core/security-transport-encryption/, 2020.

[63] M El-Mehalawi and R Allen Miller. A database system of mechanical components based on geometric and topological similarity. Part I: Representation. *Computer-Aided Design*, 35(1):83–94, 2003. ISSN 00104485. doi: 10.1016/S0010-4485(01)00177-4.

[64] M El-Mehalawi and R Allen Miller. A database system of mechanical components based on geometric and topological similarity. Part II: Indexing, retrieval, matching, and similarity assessment. *Computer-Aided Design*, 35(1): 95–105, 2003. ISSN 00104485. doi: 10.1016/S0010-4485(01)00178-6.

[65] S Jeong, J Byun, D Kim, H Sohn, I. H Bae, and K. H Law. A data management infrastructure for bridge monitoring. In J. P Lynch, editor, *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, page 94350P, San Diego, California, United States, 2015. doi: 10.1117/12.2177109.

[66] R Volk, J Stengel, and F Schultmann. Building Information Modeling (BIM) for existing buildings — Literature review and future needs. *Automation in Construction*, 38:109–127, 2014. ISSN 09265805. doi: 10.1016/j.autcon.2013.10.023.

[67] M de Oliveira, R Nascimento, and D Brandao. A new real-time SHM system embedded on raspberry pi. In *European Workshop on Structural Health Monitoring*, pages 386–395. Springer, 2022.

[68] L Atzori, A Iera, and G Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010. ISSN 13891286. doi: 10.1016/j.comnet.2010.05.010.

[69] S. J Johnston, M Apetroaie-Cristea, M Scott, and S. J Cox. Applicability of commodity, low cost, single board computers for Internet of Things devices. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 141–146, Reston, VA, USA, 2016. IEEE. ISBN 978-1-5090-4130-5. doi: 10.1109/WF-IoT.2016.7845414.

[70] K. Y Koo, N de Battista, and J. M. W Brownjohn. SHM data management system using MySQL database with Matlab and web interfaces. 2011.

[71] M Valinejadshoubi, A Bagchi, and O Moselhi. Development of a BIM-Based Data Management System for Structural Health Monitoring with Application to Modular Buildings: Case Study. *Journal of Computing in Civil Engineering*, 33(3):05019003, 2019. ISSN 0887-3801, 1943-5487. doi: 10.1061/(ASCE)CP.1943-5487.0000826.

[72] J Rio, B Ferreira, and J Poças-Martins. Expansion of IFC model with structural sensors. *Informes de la Construcción*, 65(530):219–228, 2013. ISSN 1988-3234, 0020-0883. doi: 10.3989/ic.12.043.

[73] M O'Shea and J Murphy. Design of a BIM Integrated Structural Health Monitoring System for a Historic Offshore Lighthouse. *Buildings*, 10(7):131, 2020. ISSN 2075-5309. doi: 10.3390/buildings10070131.

[74] A Mita, H Sato, and H Kameda. Platform for structural health monitoring of buildings utilizing smart sensors and advanced diagnosis tools. *Structural Control and Health Monitoring*, 17(7):795–807, 2010. ISSN 15452255. doi: 10.1002/stc.399.

[75] M Pregnolato. Bridge safety is not for granted – A novel approach to bridge management. *Engineering Structures*, 196:109193, 2019. ISSN 01410296. doi: 10.1016/j.engstruct.2019.05.035.

[76] N Testoni, C Aguzzi, V Arditi, F Zonzini, L De Marchi, A Marzani, and T. S Cinotti. A Sensor Network with Embedded Data Processing and Data-to-Cloud Capabilities for Vibration-Based Real-Time SHM. *Journal of Sensors*, 2018:1–12, 2018. ISSN 1687-725X, 1687-7268. doi: 10.1155/2018/2107679.

[77] M McVay, M Hoit, E Hughes, T Nguyen, and P Lai. Development of a Web Based Design, and Construction Bridge Substructure Database.

[78] J Arliansyah, Y Utama, M Arlini Wijayanti, R Gusti, and Arifianto. Analysis and design of road and bridge infrastructure database using online system. *MATEC Web of Conferences*, 138:07011, 2017. ISSN 2261-236X. doi: 10.1051/matecconf/201713807011.

[79] E. F Codd. *Further Normalization of the Data Base Relational Model*. Courant Computer Science Symposia 6, Data Base Systems. Prentice-Hall, 1972.

[80] D Crawford. Technical correspondence. page 2.

[81] MongoDB, Inc. About Us - MongoDB. https://www.mongodb.com/company, 2020.

[82] T Bray. RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format. https://tools.ietf.org/pdf/rfc7159.pdf, 2020.

[83] T Bray. RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format. https://tools.ietf.org/pdf/rfc8259.pdf, 2020.

[84] MongoDB, Inc. Role-Based Access Control - MongoDB Manual. https://docs.mongodb.com/manual/core/authorization/, 2020.

[85] MongoDB, Inc. Storage Engines - MongoDB Manual. https://docs.mongodb.com/manual/core/storage-engines/, 2020.

[86] MongoDB, Inc. WiredTiger Storage Engine - MongoDB Manual. https://docs.mongodb.com/manual/core/wiredtiger/\#document-level-concurrency, 2020.

[87] MongoDB, Inc. Indexes - MongoDB Manual. https://docs.mongodb.com/manual/indexes/index.html, 2020.

[88] MongoDB, Inc. Replication - MongoDB Manual. https://docs.mongodb.com/manual/replication/, 2020.

[89] MongoDB, Inc. Sharding - MongoDB Manual. https://docs.mongodb.com/manual/sharding/index.html, 2020.

[90] C. M Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, reprinted edition, 2010. ISBN 978-0-19-853864-6.

[91] F Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 1939-1471, 0033-295X. doi: 10.1037/h0042519.

[92] D Bacciu, F Errica, A Micheli, and M Podda. A Gentle Introduction to Deep Learning for Graphs. *Neural Networks*, 129:203–221, 2020. ISSN 08936080. doi: 10.1016/j.neunet.2020.06.006.

[93] Y Li, C Gu, T Dullien, O Vinyals, and P Kohli. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. *arXiv:1904.12787 [cs, stat]*, 2019.

[94] D. P Kingma and J Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[95] M Degener and M Hermes. Ground vibration test and finite element analysis of the GARTEUR SM-AG19 testbed. Technical report, 1996.

[96] E Balmès and J. R Wright. Garteur Group on Ground Vibration Testing: Results From the Test of a Single Structure by 12 Laboratories in Europe. In *Volume 1A: 16th Biennial Conference on Mechanical Vibration and Noise*, page V01AT03A004, Sacramento, California, USA, 1997. American Society of Mechanical Engineers. ISBN 978-0-7918-8040-1. doi: 10.1115/DETC97/VIB-4255.

[97] M Link and M Friswell. Working Group 1: Generation of validation structure dynamic modles—results of a benchmark study utilising the GARTEUR SM-AG19 test-bed. *Mechanical Systems and Signal Processing*, 17(1):9–20, 2003. ISSN 08883270. doi: 10.1006/mssp.2002.1534.

[98] Laboratory for Verification and Validation. https://lvv.ac.uk/.

[99] H Fraser-Mitchell. The Hawk Story. *Journal of Aeronautical History*, 2013/01, 2013.

[100] M Rolfe. BAE Systems Hawk Blueprint. https://drawingdatabase.com/bae-systems-hawk/.

[101] R. A Force. Hawk T2 — Royal Air Force. https://www.raf.mod.uk/aircraft/hawk-t2/.