Mechanical Systems and Signal Processing 223 (2025) 111602

Contents lists available at ScienceDirect

Mechanical Systems and Signal Processing

journal homepage: www.elsevier.com/locate/ymssp

Foundations of population-based SHM, Part V: Network, framework and database

Daniel S. Brennan *, Julian Gosliga, Elizabeth J. Cross, Keith Worden

Dynamics Research Group, Department of Mechanical Engineering, University of Sheffield, Mappin Street, Sheffield, S1 3JD, UK

ARTICLE INFO

Communicated by J.E. Mottershead

Keywords:
Population-based Structural Health Monitoring (PBSHM)
Framework
Database
Irreducible Element (IE) model

ABSTRACT

This paper represents the final part of a sequence on the foundations of *Population-Based Structural Health Monitoring* (PBSHM). The previous papers presented detailed analyses of how PBSHM can extend the functionality of 'classical' single-structure SHM by leveraging information and data available across a population of structures. Two main concepts were critical in establishing the basic methodology. In the first place, it was assumed that the uplift from considering a population would only be truly present if a structure of interest could be shown to be 'similar' in some sense to at least one other structure in the population. This requirement of similarity demanded the development of a principled means of measuring the degree of said similarity; this was accomplished by the proposal and design of *Irreducible Element* (IE) and *Attributed Graph* (AG) models of structures, which naturally resided in a metric space. The second key concept in PBSHM was the idea of moving inferences between structures, and the natural methodology for this was established as *transfer learning*. This final paper is concerned with showing how the main ideas of PBSHM can be embedded in software; in fact, this proved to be non-trivial and not a simple matter of programming. In order to embed the concepts of PBSHM within a computational framework, a number of new concepts needed to be developed – the *network*, *framework* and *database* – and these ideas are introduced within the paper. Furthermore, the development of the computational architecture revealed a number of limitations inherent in the previously-proposed base PBSHM ideas, notably in the conception of the IE model. This paper therefore provides a radical reformulation of the IE-model concept, presenting it in a new form appropriate to embedding in software. The development of new theoretical constructs proved to be so entwined with the software development issues that they are not easily separable, so new concepts and software implementation are both presented here. To avoid interrupting the narrative flow of the conceptual material, the details of many of the software 'objects' are presented in a substantial appendix.

1. Introduction

This paper is the fifth in a series addressing the foundations of Population-based Structural Health Monitoring (PBSHM). Traditionally, Structural Health Monitoring (SHM) [1], has involved implementing a damage-detection strategy for a single structure for which the health-state is to be determined. PBSHM expands on this process [2–5] by monitoring multiple structures — (*the population*) — with the goal of gaining additional insights into the health of a given structure when using population data, compared to those available from the single structure's data. As discussed in the earlier papers in this series, to achieve the goals of PBSHM, two

* Corresponding author.
E-mail address: d.s.brennan@sheffield.ac.uk (D.S. Brennan).

https://doi.org/10.1016/j.ymssp.2024.111602
Received 31 October 2023; Received in revised form 8 April 2024; Accepted 3 June 2024
Available online 21 August 2024

basic problems need to be addressed: finding the similarity of structures (*or components of the structure*), and transferring knowledge learnt across the population.

The second part in this series [3], explored the problem of finding the geometrical similarities between theoretical structure datasets. When comparing the composition of components across multiple structures, a shared description was required to facilitate equal identification of similarity. Irreducible Element (IE) models were the vehicle introduced by Gosliga et al. to facilitate this shared description of structures within PBSHM.

The fourth part in this series [5], explored the use of a fibre-bundle to facilitate a shared space in which knowledge could be transferred across structures. From the work in both the second and fourth part of this series, it has become clear that PBSHM by its very nature, has an underlying obstacle of data transparency and compatibility because of the vast expanse of structures targetted. For PBSHM to reach beyond a theoretical concept, a shared domain for PBSHM data is required. This shared domain must not only provide a common standard for compatibilities' sake, but also facilitate the domain in which PBSHM computations reside.

The purpose of this paper within the series, is to provide the underlying technical implementations for the aforementioned shared data domain in PBSHM. The motivation behind why PBSHM requires its own shared data domain is simple. Inherently, every structure has its own set of labels, descriptions, attributes, formats, and reasons for why there are current data on the structure. To adopt a new structure into the PBSHM network, entails modifying any relevant PBSHM algorithms to understand the data of the structure compared to the rest of the population. If instead, a shared standard can be proposed for PBSHM, data only need to be translated into this standard and the focus of PBSHM can instead rest on the development of new PBSHM-derived algorithms.

This paper introduces a PBSHM shared-data domain 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). Furthermore, in PBSHM, establishing the similarity between two structures is critical to enabling positive transfer of learnt knowledge from one structure to another. As such, a large proportion of this paper is dedicated to expanding the current definitions of an Irreducible Element (IE) model to facilitate IE models becoming the standardised methods by which structures are described within PBSHM.

The previously referenced work in part two of this series [3] presented the minimal essence of an IE model, before moving onto the comparisons of models via the Attributed Graph (AG). The introduced IE model lacked both the firm definitions and formal specification required for a standardised language to emerge. The IE model concept was also void of the required domain knowledge to understand certain structural scenarios within the model. A fresh look at the IE model abstraction was necessitated; therefore, this paper provides the missing definitions, formal specification, and expansion of IE model theory to materialise the desired standardised method for describing structure within PBSHM. It is important to note that whilst the language of an IE model has been modified since the part two [3] paper, the introduced conversion between IE model and AG is still valid and is utilised within this paper.

Part of the reason this paper has arrived much later than its sisters is because the development has involved a complex interplay between fundamental research on PBSHM concepts and their implementation in software. This interplay means that the paper weaves together the definition and extension of PBSHM concepts with the details of their embodiment within code. In the interest of clarity in the narrative structure, the technical tables required for the interested reader to produce their own IE-model of a structure are extracted from the main body of this paper and included within Appendix.

Section 2 outlines the current state of a shared domain with PBSHM. Section 3 sets out the definitions of the *network*, *framework*, and *database*. Section 4 explains the necessary changes to Irreducible Element (IE) models to enable them to become a standardised language for describing a structure within PBSHM. Section 5 describes the technical implementations constructed for the *database* and *framework*. Section 6 shows the *framework* computing similarity scores for structures and outlines the future work required.

Throughout this paper a common notation is used to denote different data; *named object* will reference a generalised object name, [type of named object] will reference a type of *named object* and 'value' will reference an absolute value of a property within said object.

2. Background

Structural Health Monitoring (SHM) [1,6], 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, SHM aims to detect damage within a structure. There are two widely-explored approaches within the SHM community: model-based and data-based. A model-based approach aims to take the existing knowledge of the dynamic behaviour of a physical structure and encapsulate this knowledge in the form of a model — commonly a Finite Element (FE) model. The desire is that the model will respond to an input in the same manner as the physical structure. A data-based approach instead operates from the premise that there is no pre-existing dynamic knowledge of a structure and instead builds a statistical model from sensor data based upon known 'healthy' states of the structure. Raw sensor data will seldom provide a direct indication of damage present within a structure, instead raw data are processed into damage sensitive features — features which are likely to change in value (e.g. natural frequencies), if damage becomes present in the structure.

Both of the aforementioned approaches suffer from a problem of data availability. With a model-based approach, is there enough existing knowledge regarding the dynamic behaviour to accurately produce a model of the structure? In a data-based approach, is the structure able to be monitored to provide a 'healthy' state, and are data available representing any damage states? This is where

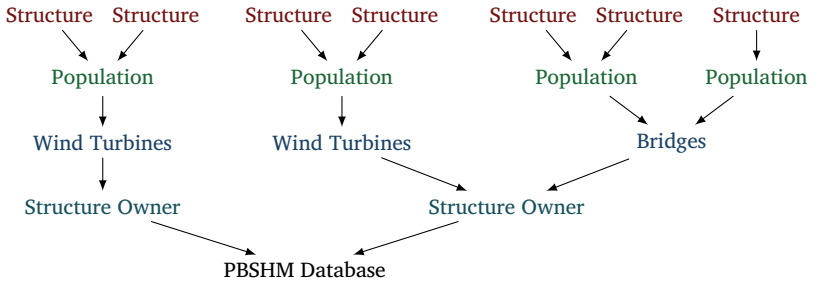


Fig. 1. Data transmission from structure owners' current SHM systems into a PBSHM database.

Population-based Structural Health Monitoring (PBSHM) [7] aims to provide a potential solution for the data-scarcity problem in SHM.

The premise of PBSHM, is that by utilising data and knowledge from a group of similar structures, additional knowledge on the health of an individual structure is established, compared to a traditional SHM approach. However, this methodology comes with its own set of problems which require addressing. In the interest of brevity, this paper will not revisit the previous problems addressed in the first four parts [2–5] of the 'foundations' series. Instead, this paper will focus on addressing the data-domain problem faced within PBSHM (see Section 3).

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 [8].

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.

There are existing standards which address these data requirements for a particular type of structure. Jeong et al. [9] discuss existing standards used within bridge lifecycle management, such as the Bridge Information Modelling (BrIM) standard. These standards provide a data structure, enabling easy comparisons; however, they lack the functionality to extend the specification to include different structure types and associated data types/units. This issue often means that bespoke methodologies must be developed for every type of structure within an SHM application; not only does it make analysing and processing of PBSHM data costly, it puts unnecessary barriers in place for adoption of PBSHM technologies.

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.

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 Fig. 1)

2.1. Database

Databases, at their core, are simply a technique for storing data in a structured manner. Codd [10] introduced this idea in 1970 and it has since become the foundation of modern *Relational Database Management Systems* (RDBMS). However, since the

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

Fig. 2. A relational example showing a "structure-1" structure containing four sensors within an RDBMS.

introduction of this principle, the constraint of data being structured/relational has caused a new breed of databases to be born. *Not only Structured Query Languages* (NoSQLs) are a family of database methodologies that aim to solve the structured data problem by enabling schema-free data storage. *Structured Query Language* (SQL) is used to manage data inside an RDBMS, and as such, NoSQL databases are designed without these inherent constraints. There are *vertical* and *horizontal* scalability concerns as a result of the nature of an RDBMS. *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. [11] and Zafar et al. [12] review these concerns in conjunction with detailing differences between RDBMS and NoSQL. Gandini et al. [13] outline some performance-planning considerations when using a NoSQL database.

It should be noted that some security concerns highlighted by Nance et al. [11], regarding encryption at rest,[1] authentication in a *sharded*[2] environment and client communication encryption have been addressed [14–17] since the publication of these papers.

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 set forth in this paper.

2.1.1. RDBMS

RDBMS, in its most simplified version, stores data in the format of tables and columns (see Fig. 2). Relationships within are created via *primary keys* and *foreign keys*. Primary keys are the method 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 [10] introduced what is now known as first normal form (1NF) and later introduced second and third normal form (2NF and 3NF) [18,19]. 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 3.3 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 SQL statements are required to retrieve the state of a single structure.

2.1.2. 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 nodes and edges to store data; they function via the same rules as graph theory. 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. [12] discuss the merits and drawbacks of each type of NoSQL database in additional detail.

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 Fig. 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 algorithms inside the PBSHM framework first. 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.

[1] Encrypted Storage Engines are available as of MongoDB 3.2 Enterprise [14,15].

[2] *Sharding* is a method of splitting data across multiple database instances.

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

Fig. 3. An embedded example showing a "structure-1" structure containing four sensors within a NoSQL Document database.

3. Network, framework, and database

One of the main aims of PBSHM is to facilitate transfer of information between similar structures in order to improve diagnostic power on individuals with little or no damage-state data. This aim breaks down into two fundamental objectives; finding which structures (*or components of structures*) are similar [3], and transferring learnt knowledge between similar structures [4]. Effective implementation of the aim requires a shared domain for data residency on which the comparisons of structures and transfers of knowledge can be made. Of course, the aforementioned comparisons require a representation of structures amenable to comparisons and a means of storing these representations; this will be the PBSHM *database*.

In addition to the database itself, further infrastructure is needed in order to support the comparison and transfer methods; this will need to account for the specific complexities inherent in data-based SHM like the accommodation of environmental states and operational variations which, if not dealt with, can confuse diagnostic algorithms. A full PBSHM system needs to account for the environment of the structures of interest and operational choices like sensor distributions. Beyond the structural models, one needs a representation of the reality in which the structures are embedded. The extension from the structural models to the *reality model* (see Section 6.1) will ultimately be accomplished by extending the *database* to a complex network (the *network*), which is itself embedded in a learning and inference framework (the *framework*). This paper will give outlines of the *network* and *framework* architectures; however, detailed development and description will be postponed until later publications following ongoing research. The detailed technical specification of the *database* and the required representation of structures within, will be the main subject of this paper.

Earlier papers in this series outlined the basic concepts regarding structural representation; Gosliga et al. [3] provided the basic definitions of Irreducible Element (IE) models and their associated graphs, Gardner et al. [4] described how transfer between specific structural feature spaces could be achieved, and Tsialiamanis et al. [5] gave a geometrical overview of PBSHM which could form the basis of methodology development.

None of these publications addressed the gritty technical issue of how a shared-data domain could be designed to hold structure representations, which would be forward compatible with the many developments that establishing the *network* and *framework* would require. The main aim of this paper is to present the database (the shared-data domain) in some detail, while describing enough of the *network* and *framework* requirements to instil confidence that the database will be forward compatible. Development of the database has required a re-evaluation of the principles of the IE-models themselves and has driven new research, which is reported here.

3.1. 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 3.3) reside. Each relationship within the network will have been established via an algorithm implemented in the PBSHM Framework (see Section 3.2). 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 PBSHM Database, the PBSHM 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 PBSHM database at the given time.

Fig. 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 an edge to a structure node, represents a piece of data present within the PBSHM database which establishes the current known state of the given structure. The dotted edges between each of the structure nodes, represent a relationship established by the similarity metrics available within the PBSHM Framework. Within the fullness of time, the PBSHM 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 paper will focus on the implementations required to support *IE models* [3] and *channel data*.

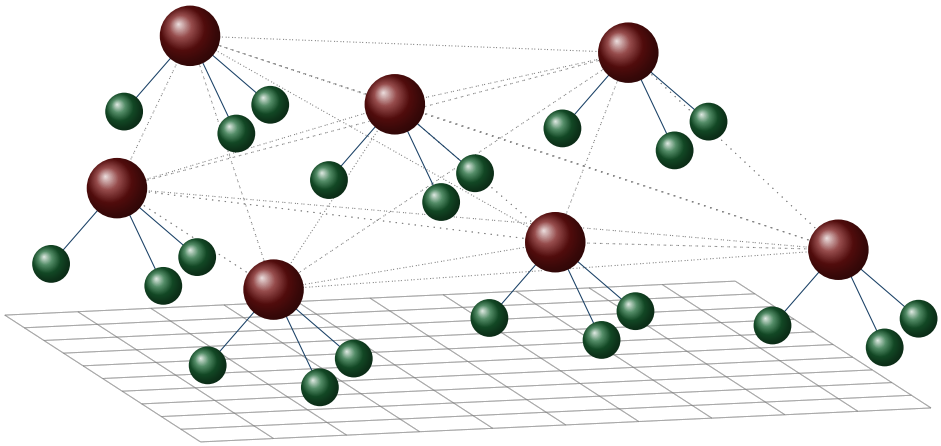


Fig. 4. A visual representation of the network of structures stored within the PBSHM Framework and PBSHM database. The large red nodes represent the presence of a structure within the network, the green nodes represent the data stored within the PBSHM database which depict the current known state of the given structure. The dotted edges between the structure nodes represent the relationships present between structures, because of similarities metrics generated by the PBSHM Framework.

3.2. 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 3.3). 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.

Whilst the ideal scenario for the PBSHM framework may be to have a singular centralised system which encapsulates all the available PBSHM data, the authors realise 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. Fig. 5 depicts the overall hierarchy of how the modules, core system and database interact together, to fulfil the desired functionality of PBSHM.

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 modules will be decentralised from the authors of this paper, 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 when the population size is N or 1. 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 3). 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.

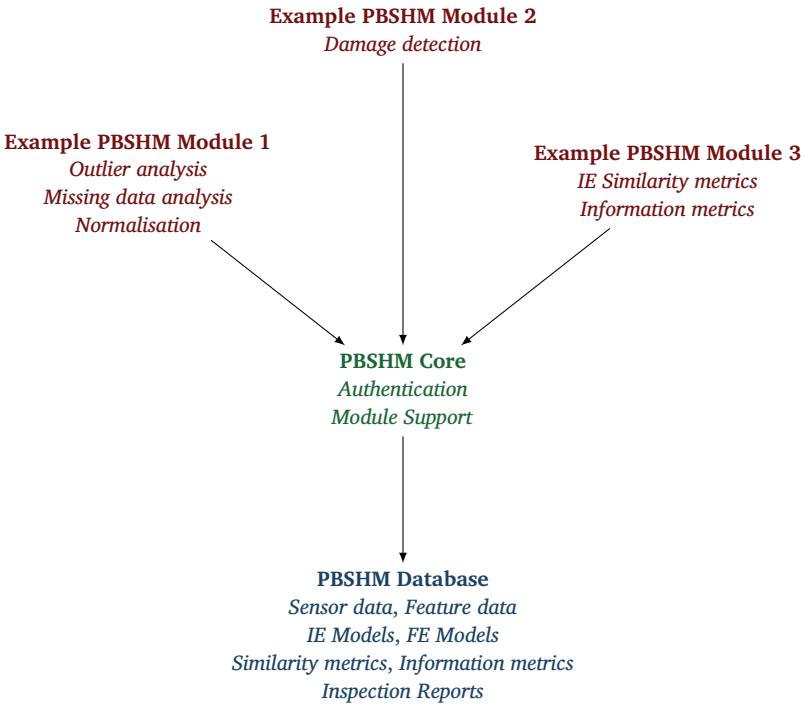


Fig. 5. The hierarchical structure of the PBSHM Framework; PBSHM Modules interacting with the PBSHM Core and subsequently the PBSHM Core then interacting with the PBSHM Database to fulfil the requests from the PBSHM Modules.

3.3. The database

Underpinning the ecosystem described within this paper, 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. IE model

The notion of an Irreducible Element (IE) model was introduced by Gosliga et al. [3] in Part II of this 'Foundations of Population-based Structural Health Monitoring' series. An IE model is the vehicle used within PBSHM to describe significant structural components of the system and the interactions between said components. This model comprises the abstract representation required to evaluate the similarity of heterogeneous structures within the network. 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.

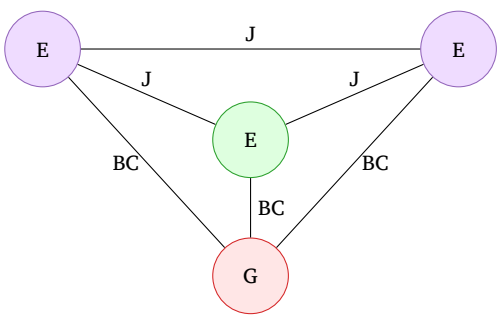
An IE model describes the structure in question; however, an IE model cannot be directly inserted into the network of structures as the network exists in a graph space. Subsequently, an IE model representation is transformed into an Attributed Graph (AG), via each component within the model becoming a node 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 node/edge.

Gosliga et al. [20] 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 a 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.

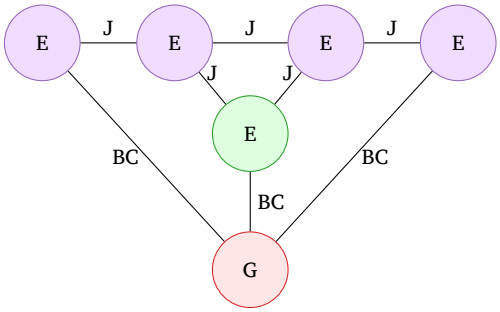
The objective of the proposed changes within this paper are to expand the core concepts of an IE model and subsequently, the language used to describe the structure being modelled. The aim is that by making these modifications, additional engineering knowledge and design choices regarding the structure 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.

Fig. 6 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 Fig. 6(a)) 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.

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



(b) An Attributed Graph representation of Fig. 6(a) when using only one element per span.



(c) An Attributed Graph representation of Fig. 6(a) using two elements per span because of sensor localisation or damage localisation requirements.

Fig. 6. 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.

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 Fig. 6(b) and 6(c)). 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.

The aspiration of the PBSHM Framework 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 PBSHM Framework 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, an expanded language and concept of IE models is proposed. To achieve this extended concept and language of IE models, the theory put forth by Gosliga et al. (see Fig. 7) is expanded upon, restructured and unified into the PBSHM Schema enabling a standardised representation of structures within PBSHM and the network of structures.

The remainder of this section – in conjunction with the associated Appendix B – provide the details of the expanded IE-model language and concept, including details on nested objects and terminology within the model. A brief overview of the hierarchical restructuring taken place between the original concept of IE-model language and the version included within this paper, can be viewed in Figs. 7 and 9. Furthermore, an example of how to use the IE-model language included within this paper, is provided by Brennan et al. [21] using a real-world Hawk T.Mk1 aeroplane.

4.1. 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 node and each *relationship* becomes an edge. Any properties associated with a *root*

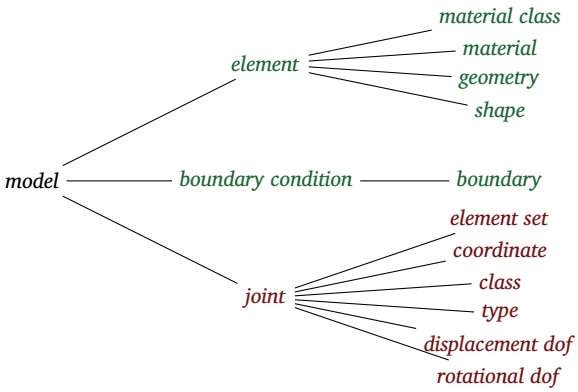


Fig. 7. The original hierarchical data models used for describing an Irreducible Element model as described by Gosliga et al. [3,20].

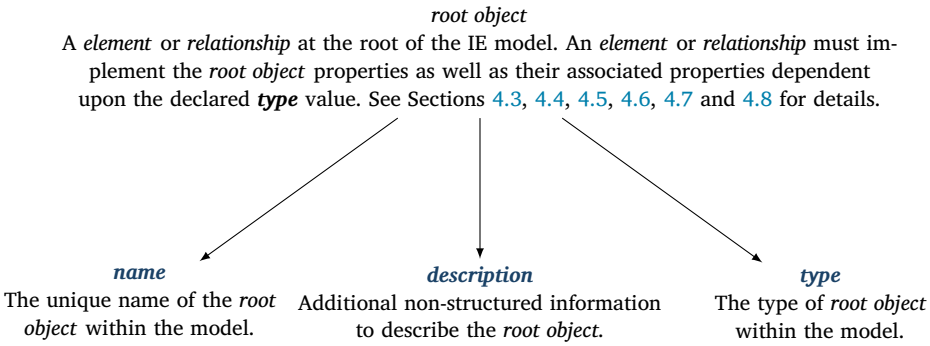


Fig. 8. The properties for a *root object* in an Irreducible Element model.

object – an *element* or a *relationship* – become embedded attributes on the associated node/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 Fig. 8 for details on the properties available for all *root object*s. In the interest of clarity, further documentation within this paper will use the annotation of [type] *object*, where [type] refers to the sub category/type of the named object: *object*.

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.

4.2. 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 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.

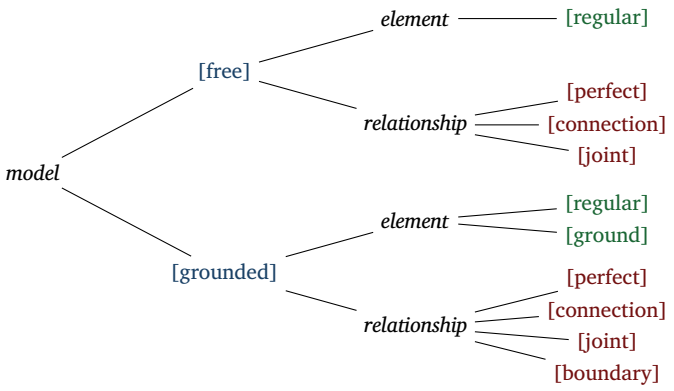


Fig. 9. The hierarchical data models used in the standardised version of Irreducible Element models. The above diagram depicts which *element* and *relationship* types are available for a given *free* or *grounded* model.

External systems are represented within a [grounded] model via [ground] *element*s (see Section 4.3) and [boundary] *relationship*s (see Section 4.8), 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. Fig. 9 depicts the permitted *element* and *relationship* types for the given [grounded] and [free] models.

4.3. 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 an interaction with the earth as the skybridge's interactions with the earth are via the skyscrapers.

In the currently-proposed solution within this paper, a [ground] *element* (see Table 22 for a list of relevant properties) does not have any additional *object* properties over the base *root object* (See Fig. 8); 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.

4.4. 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 4.3) 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 4.4.1–4.4.4 respectively). Fig. 10 depicts the hierarchical layout of a [regular] *element* and the corresponding *child objects*.

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.

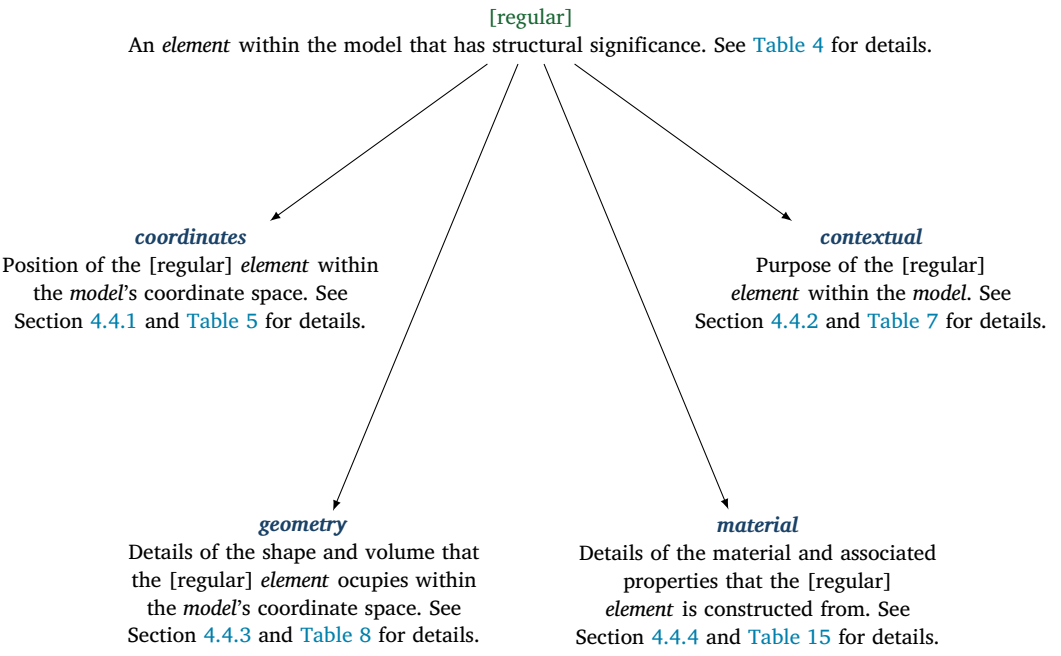


Fig. 10. The hierarchical layout of child objects within a [regular] *element*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table 4.

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.

4.4.1. Coordinates

The *coordinates* object within a [regular] *element* captures data on the element position in the 3D space of the IE model. Although provision of position data is optimal, it is recognised that this might not always be possible; as such, a *coordinates* object for a [regular] *element* is not mandatory. Fig. 11 displays the hierarchical properties within a *coordinates* object and Table 5 gives a full list of all the relevant properties.

In terms of element position, the current proposal only implements global coordinates. However, consistent with organic data design, room is left for later specification of local (element-wise) coordinate systems, without invalidating the current conventions. Whilst it is possible to provide both the translational and rotational information within the global coordinate space, only the translational information for the *coordinates* object is currently mandatory.

For clarity, the global coordinate space used within IE models has the origin O ($x = 0$, $y = 0$ and $z = 0$) in the bottom-left corner of the model. The x-axis is in the horizontal direction from left ($x = 0$) to right ($x = n$), the z-axis is in the vertical direction from bottom ($z = 0$) to top ($z = n$) and the y-axis is in depth direction from near ($y = 0$) to far ($y = n$) (see Fig. 15(c)).

4.4.2. Context

An important departure here from the original IE-model specification, is a separation of an element's function from the spatial region it occupies; the former is specified by the *contextual* object and the latter by the *geometry* object (see Section 4.4.3). The *contextual* object embeds the purpose of the component into the IE model, and is required for a valid [regular] *element*. Fig. 12 displays the hierarchical properties within a *contextual* object and Table 7 gives a list of relevant properties.

As *contextual* object captures *function* it must be described in terms appropriate to the modellers. This function is declared via a flat ***type*** property, selected from a list of supported types. As more diverse groups of structures are considered, the allowed types can be expanded.

4.4.3. Geometry

Together with the *coordinates* object (see Section 4.4.1), the *geometry* object, captures the form of a structure. As the form is critical in specifying heterogeneous populations, the *geometry* property is mandatory for a [regular] *element*. Fig. 13 shows the hierarchical properties within a *geometry* object and Table 8 gives a list of relevant properties.

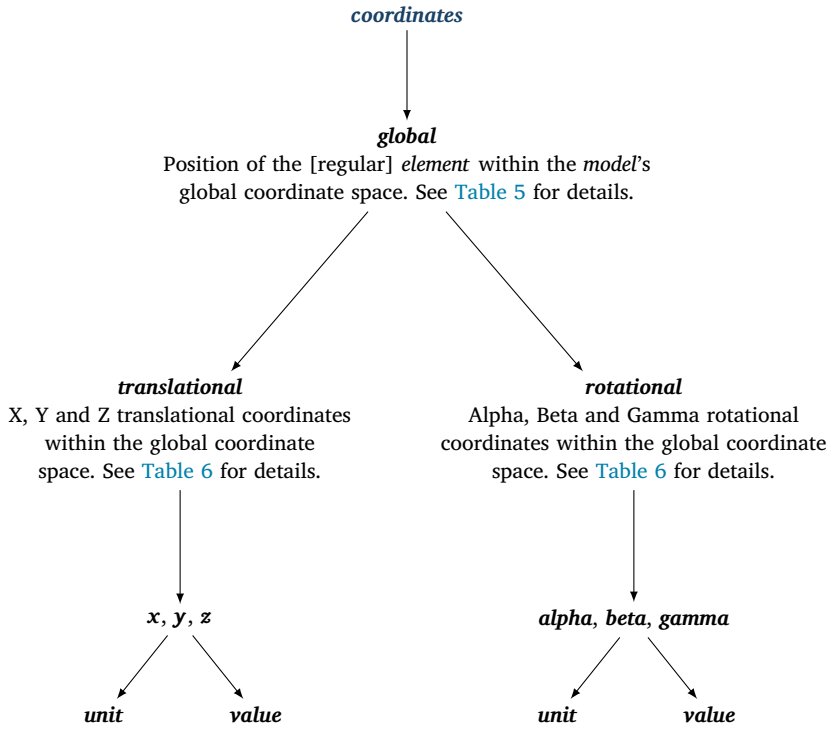


Fig. 11. The hierarchical layout of child objects within the *coordinates* section of a [regular] *element*.

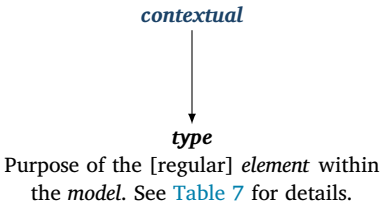


Fig. 12. The hierarchical layout of child objects within the *contextual* section of a [regular] *element*.

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; in this case a layered hierarchical forest approach is used to embed the initial knowledge. Each tree in the forest is a rooted tree, directed from the root node to the leaves. The selected type for the *geometry* object must be a complete path from a root node to its corresponding leaf node.

The reason for this hierarchical approach is clear, considering how the geometrical type may be used in determining homogeneity of structures. If one imagines two structures being compared for similarity; one [regular] *element* may be declared as a regular beam (***beam*** → ***rectangular***) in one structure and as a I-beam (***beam*** → ***i-beam***) in another model. For comparison purposes, one might say that geometry properties 'match' if the root node within the type tree is the same, e.g. if type layer 0 is ***beam***. Comparisons can be made at higher resolutions later without losing backward compatibility.

For simple shapes with common names (see Table 9), required dimensions depend on types (assuming dimensions are given). For example, for a rectangular beam (***beam*** → ***rectangular***), dimensions are not mandatory; however, if given, a minimum set of ***length***, ***width***, and ***height*** property must be declared. For more complex shapes (see Table 14), this approach will fail if a simple name does not capture proper understanding of shape.

For complex shapes, knowledge can be embedded via additional *geometry* properties; such detail being captured in supplementary type branches under the ***solid*** and ***shell*** root geometrical types. One example of such an extension is via the ***translateAndScale***

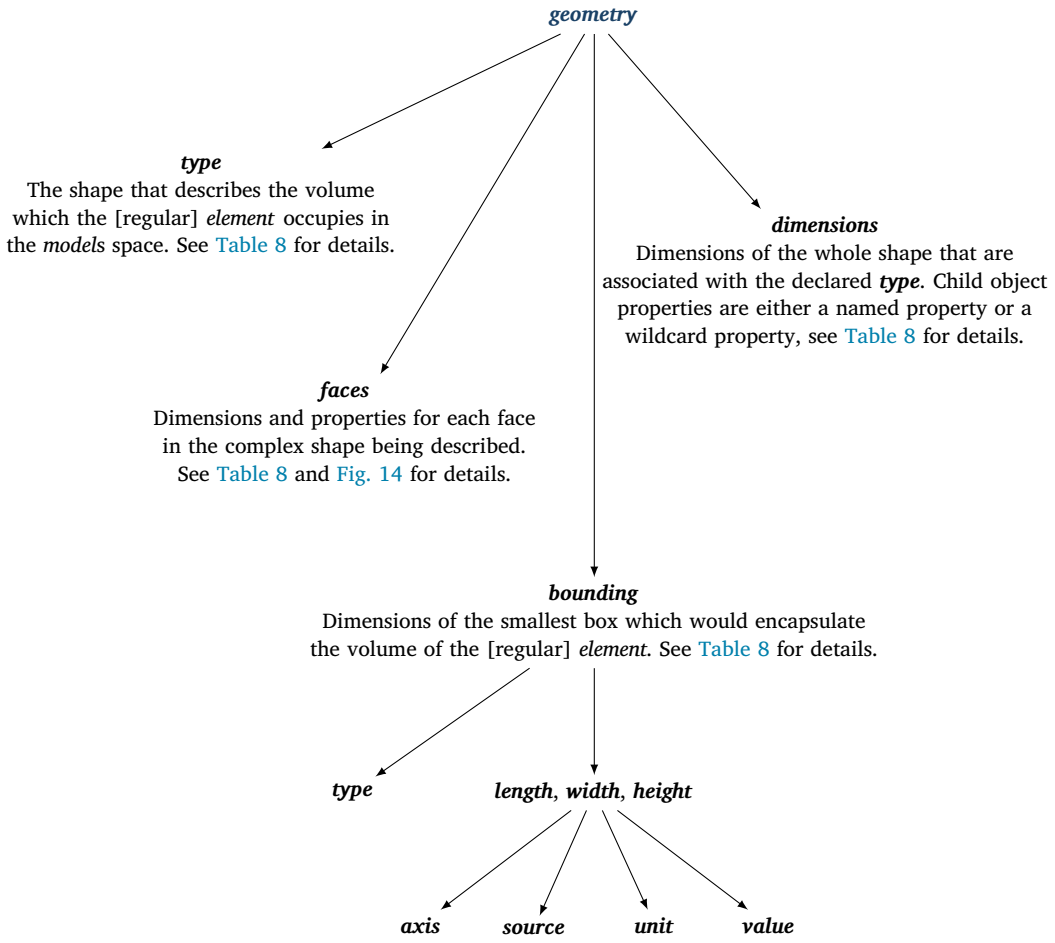


Fig. 13. The hierarchical layout of child objects within the *geometry* section of a [regular] *element* (see Fig. 14 for details on the *faces* child object).

branch type. This option applies when two opposite faces of an element have the same shape and the element is captured by the swept volume when one face is translated and scaled into the other. Figs. 15(a) and 15(b) give an example of two cuboid shapes; the first has constant cross-section and its geometry is captured by the simple specification (***solid*** → ***translate*** → ***cuboid***); the second has varying cross-section and is represented via (***solid*** → ***translateAndScale*** → ***cuboid***).

Complex/abstract shapes can be captured by the ***bounding*** and ***faces*** properties of the *geometry* object. In the future, this option will bound the shape of interest by some specified complex hull, with the ***faces*** property describing the geometrical/shape properties of the faces (*the left and right of the imaginary space in Fig. 15(b) example*). In the current implementation, only cuboidal bounding boxes are supported. Fig. 15(c) depicts the interactions between the component being modelled as a [regular] *element* and the ***bounding*** and ***faces*** properties of the *geometry* object.

4.4.4. Material

Another key component in determining similarity of structures and therefore critical in IE models is their relative material properties. The *material* object embeds the material data into a [regular] *element*. Because of the importance of material composition in comparing structures, the *material* object is mandatory. Fig. 16 displays the hierarchical properties within a *material* object and Table 15 gives list of relevant properties.

The primary method of embedding material data is via the ***type*** property (see Fig. 27). Again, the type used within the *material* object is a hierarchical forest (like the *geometry* object); however, the selected type does not have to complete a path from a root node to a leaf node, the path may terminate at any corresponding valid child node given the selected root node. Additional material properties of a [regular] *element* can be embedded within a *material* object via the ***properties*** property. In the current implementation, materials are assumed homogeneous and isotropic; in the future, general tensor specifications will be allowed. Furthermore, it will be possible to allow material properties to vary as a function of the ambient conditions of the structure, e.g. temperature.

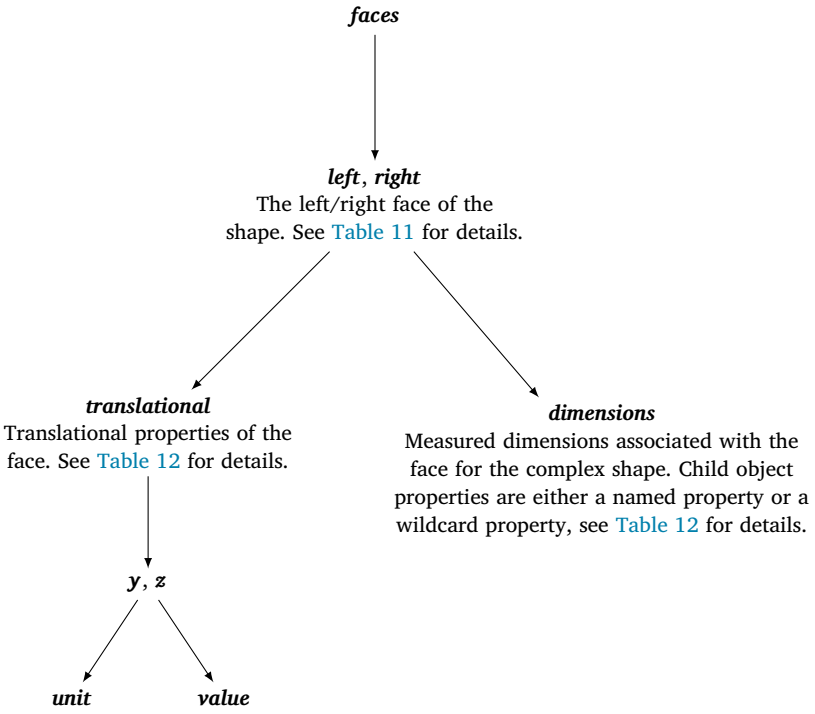


Fig. 14. The hierarchical layout of child objects within the *faces* section of *geometry* within a [regular] *element*.

For flexibility, three categories of named material properties are supported; material properties free from any associated units (see Table 17), which are deemed valid for a single value or specified range, material properties dependent upon an associated unit to correctly quantify the value (see Table 18) (again for single values or allowed ranges), and lastly, material properties which can only be considered valid with predetermined test conditions (see Table 21), and as such only accept a range of value. Where ranges are specified, the relevant variables are termed *conditional*.

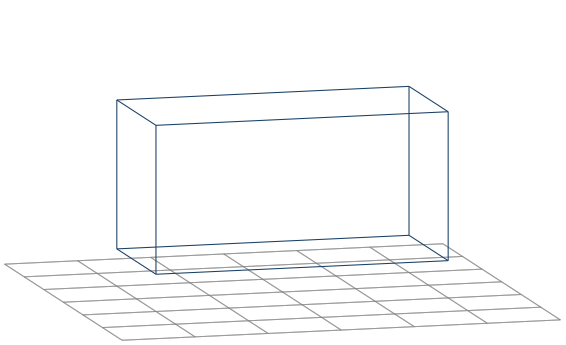
To avoid confusion, a *named* property is one which is known by the PBSHM Schema and is defined within by a set of accepted associated properties. A *wildcard* property is a property which is yet unknown by the PBSHM Schema and as such, does not constrain what the associated accepted properties are. For instance, a material property with the type set to 'yieldStrength' has an accepted list of valid ***unit*** values as defined in Table 18.

4.5. Perfect relationship

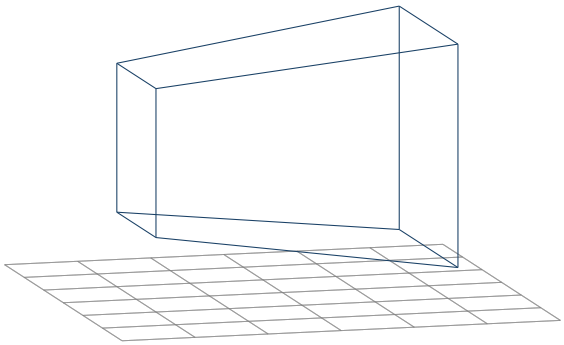
A [perfect] *relationship* represents the interaction between two – and only two – [regular] *element*s where the two *element*s in question could be considered to be the same singular *element*; segmentation only occurring to embed additional geometrical knowledge or enhanced localisation, within the 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.

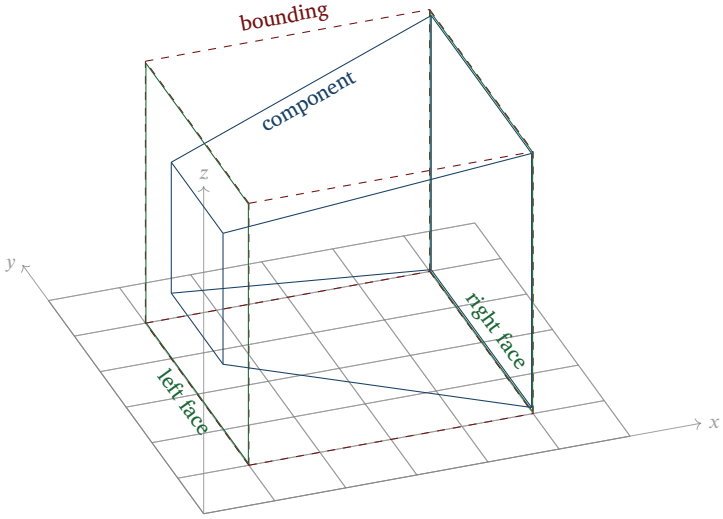
In the pursuit of embedding the form of a structure within an IE model, the *relationship*s proposed within this paper – akin to [regular] *element*s – support the inclusion of positional data via a coordinates object; depending on the number of *element*s included within a *relationship*, it determines 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.



(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***.



(c) The interaction between a component's complex geometrical shape and the associated bounding box and faces.

Fig. 15. The difference between a standard commonly-known named representation of a components geometry and a complex representation of a component geometry and the associated bounding box.

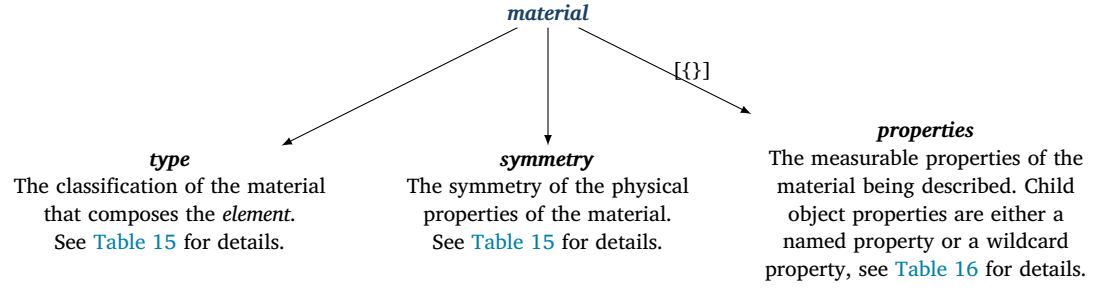


Fig. 16. The hierarchical layout of child objects within the *material* section of a [regular] *element*. A note on the [{}] symbol; this is used on the arrows to denote when the property it is pointing towards accepts an array of objects, instead of a singular value or object.

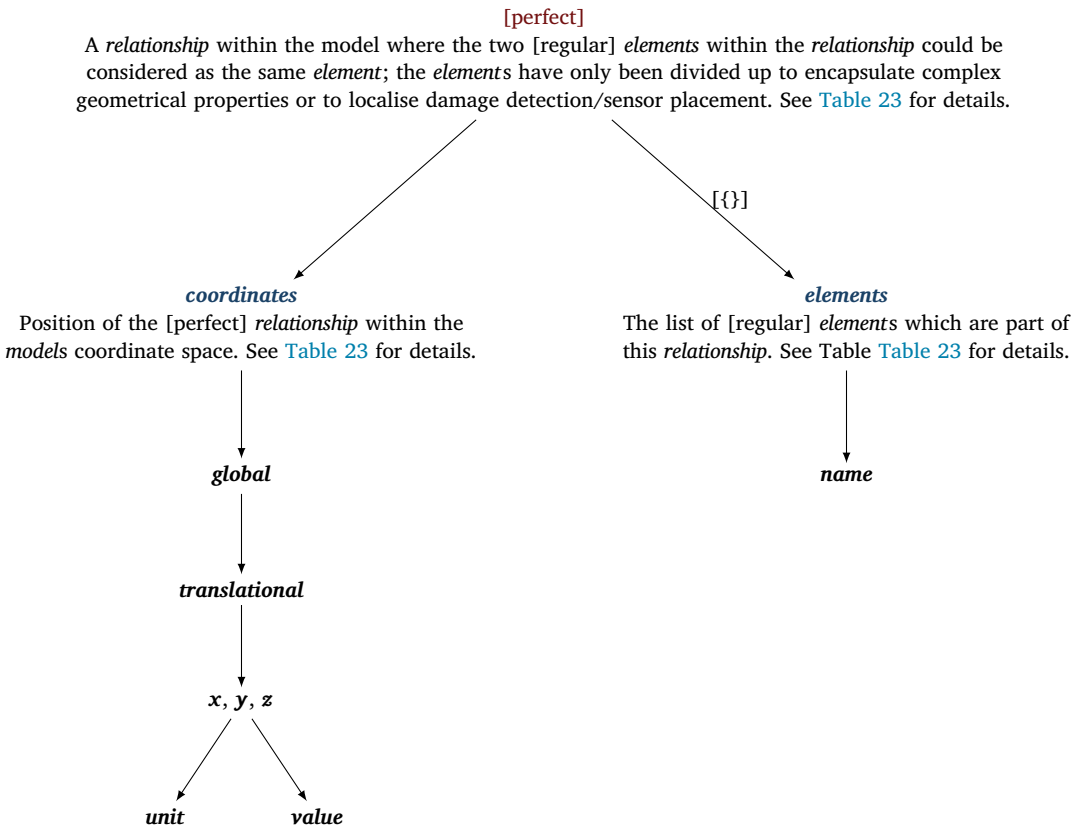


Fig. 17. The hierarchical layout of child objects within a [perfect] *relationship*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table 23.

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. Fig. 17 displays the hierarchical properties within a [perfect] *relationship* and Table 23 gives a full list of all the properties including which properties are required.

4.6. 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 4.7), without loss of knowledge within the model. The previously-omitted component is modelled by; inclusion of a new [regular] *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 Fig. 19). 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.

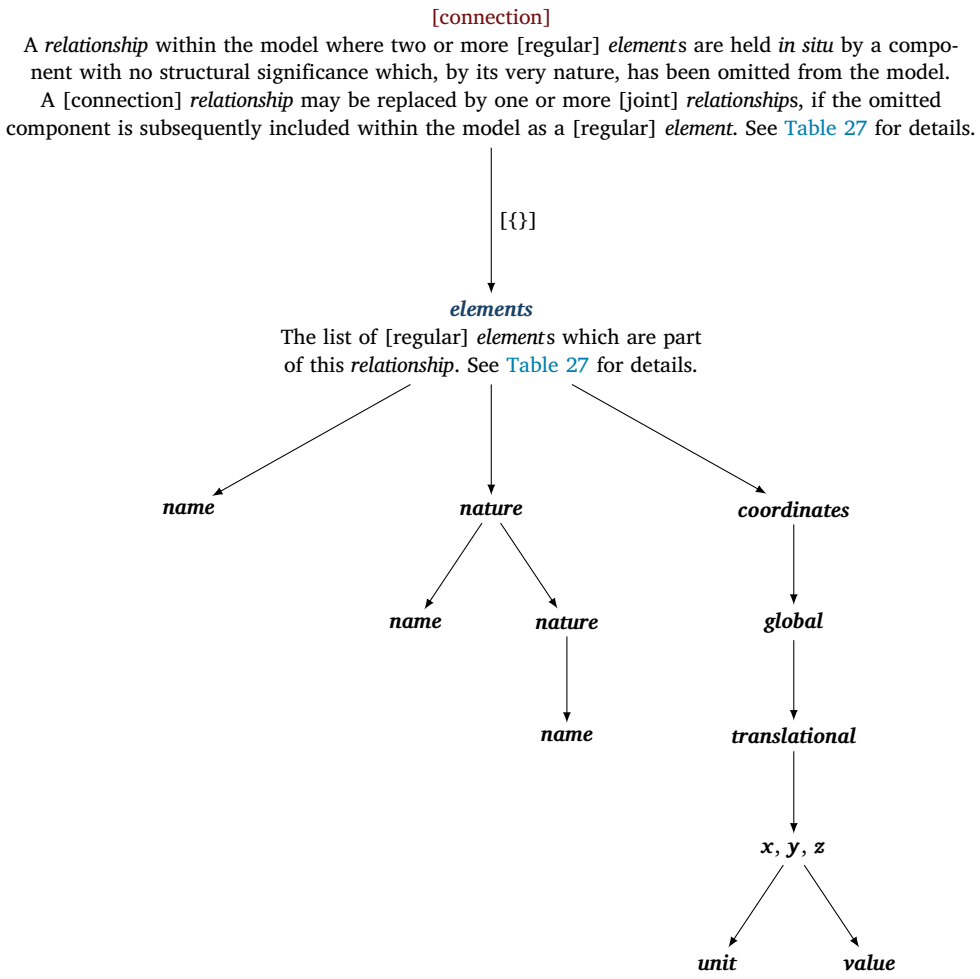


Fig. 18. The hierarchical layout of child objects within a [connection] *relationship*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table 27.

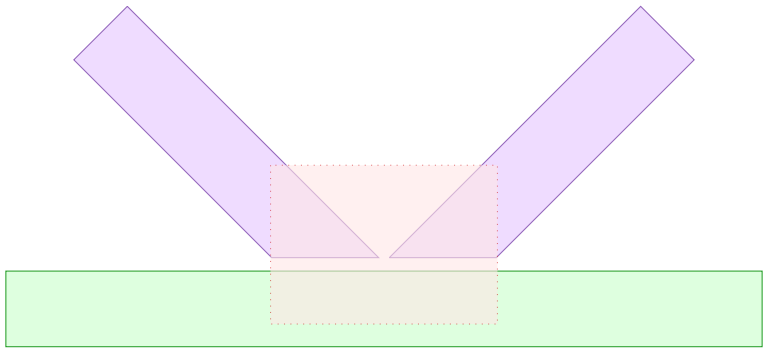


Fig. 19. An example of where a [connection] *relationship* could be used within an IE model. In this example, there are three beams all connecting together – the rectangles in purple and green – in the centre, with a face plate – the dotted red rectangle – covering the connection. This interaction can 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] *relationships*, 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*s 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*. Fig. 18 displays the hierarchical properties within a [connection] *relationship* and Table 27 gives a full list of all the properties including which properties are required.

4.7. 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 set to be [dynamic].

If one again imagines the bridge example given as part of the [connection] *relationship* (see Section 4.6). 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. Fig. 20 displays the hierarchical properties within a [joint] *relationship* and Table 29 gives a full list of all the properties including which properties are required.

4.8. 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 4.3).

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.

Fig. 21 displays the hierarchical properties within a [boundary] *relationship* and Table 23 gives a full list of all the properties including which properties are required.

4.9. Standardised representation

The beginning of this section (see Section 4) started off by discussing the problems faced when using the original IE-model language to describe structures; namely the issue that from one single structure, one may end up having multiple IE models of varying levels of granularity to describe a single structure. These variations within a model, directly impact the generated similarity metrics within the *network*; as any variation within a model will cause a disparity in the similarity for a given structure. The optimum solution is that for any given single structure, no matter what variations are present within two associated IE models, if the IE models have been generated from the same structure, they should match as identical within the *network*.

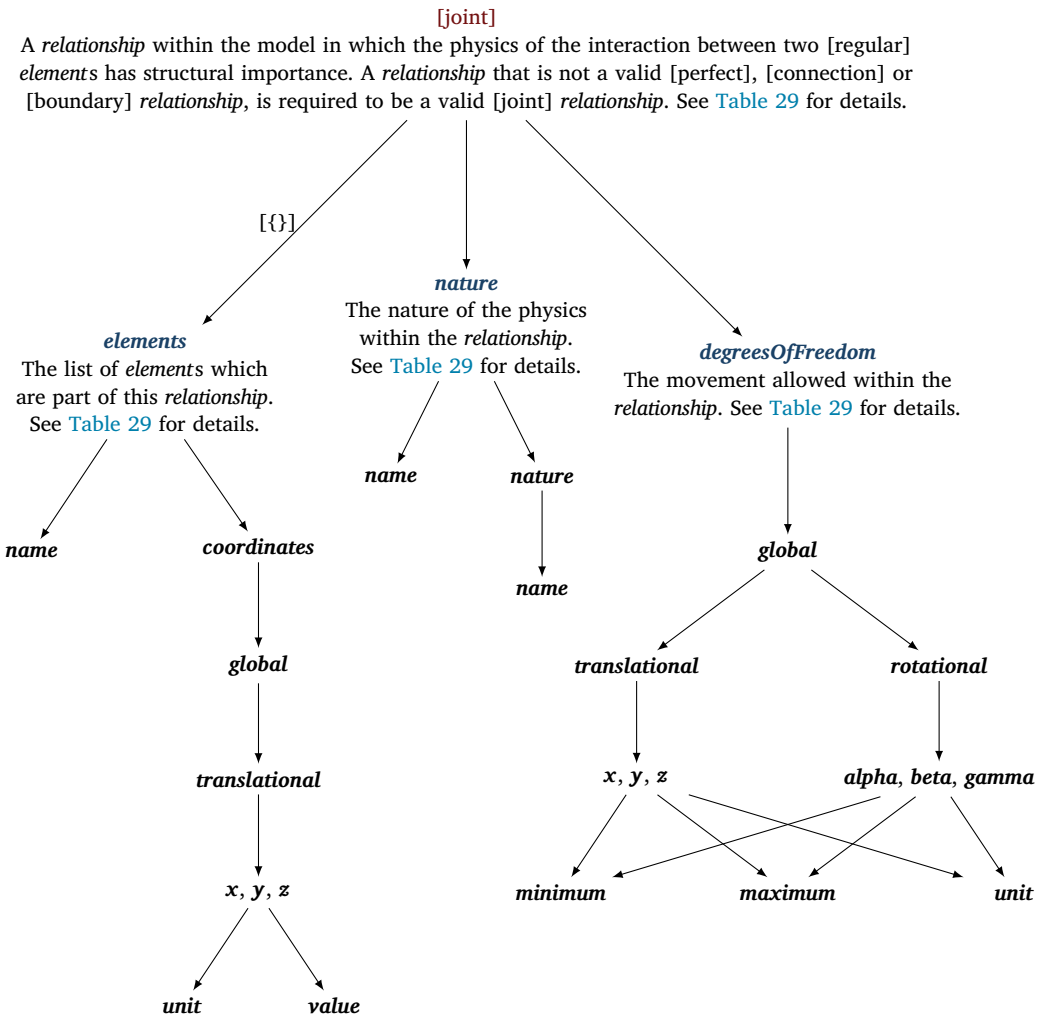


Fig. 20. The hierarchical layout of child objects within a [joint] *relationship*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table 29.

The expansion of the IE-model language included within this paper, is only solving Part One in a two-part problem to realising the aforementioned similarity within the *network*. If one can embed into the IE model, the required domain knowledge on both the structural composition and modelling decisions (Part One of the problem), one can potentially reduce the variations present within a model (Part Two of the problem) so that after variations have been removed, two IE models from the same structure will match as identical. By expanding the language, syntax, and knowledge embedded within an IE model, this paper has provided a standardised method for describing the knowledge on a structure's composition and the choices made when modelling the structure.

The structures modelled will still have variations present within the model depending upon who generates the model and the level of granularity used within the model; however, because of the expanded IE-model language included within this paper, these models now contain the knowledge to understand why variations are present and which variations can be removed in the *Canonical Form* representation (Part Two of the solution). For the interested reader, the *Canonical Form* representation of an IE model and the effect it has upon the *network* comparison space, is introduced by Brennan et al. in [22].

5. Implementation

Throughout the previous sections of this paper, the foundations, and principals of a PBSHM ecosystem have been outlined; however, the desire of this paper is to not only provide the theoretical technical components of such an ecosystem, but to provide

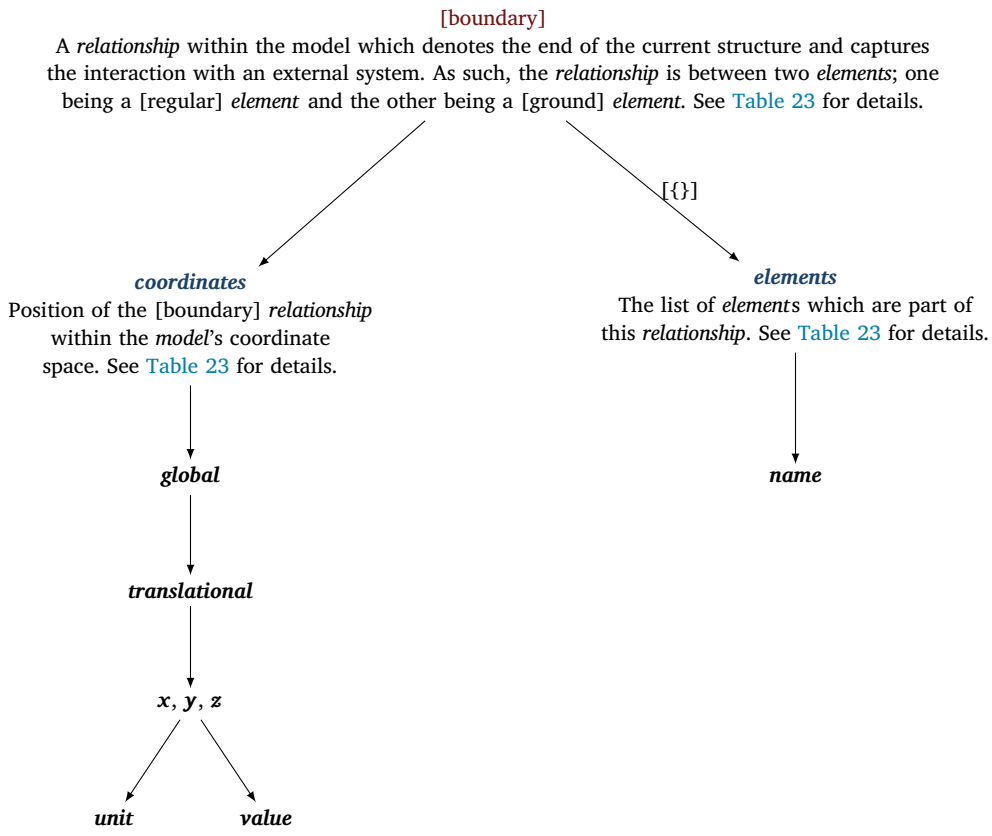


Fig. 21. The hierarchical layout of child objects within a [boundary] *relationship*. In the interest of comprehensibility, the base *root object* properties have been omitted in this figure; however, they are included in Table 23.

an initial implementation of this technology to demonstrate the potential of PBSHM. As such, the rest of this paper focuses 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 3, any implementation of the ecosystem must support both of the technical themes of PBSHM, therefore, the database and framework implementation outlined within this paper will focus on supporting the newly-proposed standardised IE model — 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.

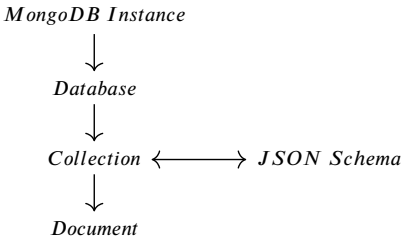
5.1. Database choice

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 [23], 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 [24,25] 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 structure data documents inside the collection.

Because of MongoDB's access controls [14,26], all users will be given read permissions 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 [14,27,28] 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* [14,29]. 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 returned 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* [14,30] and *sharding* [14,31]; however, for the purpose of this paper, they are not implemented within the PBSHM database because of the system only containing historical data that can fit within a single database node.

5.1.1. Structure

At the root of the 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 paper, additional properties have been declared for models and raw channel data; *model* → *irreducibleElement*, and *channels* respectively. Fig. 22 depicts the hierarchical objects for the *structure* object and Table 1 gives a full list of relevant properties.

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 1 MHz 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 1 GHz.[3]

5.1.2. 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 exist within this *structure* document. Fig. 23 depicts the hierarchical objects for the *channel* object. A description of each property within the *channel* entity is included in Table 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 is not 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 40.

5.1.3. 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 41.

[3] When using Javascript to interact with the PBSHM Schema, there will be a loss of accuracy 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 paper it is implemented in Javascript.

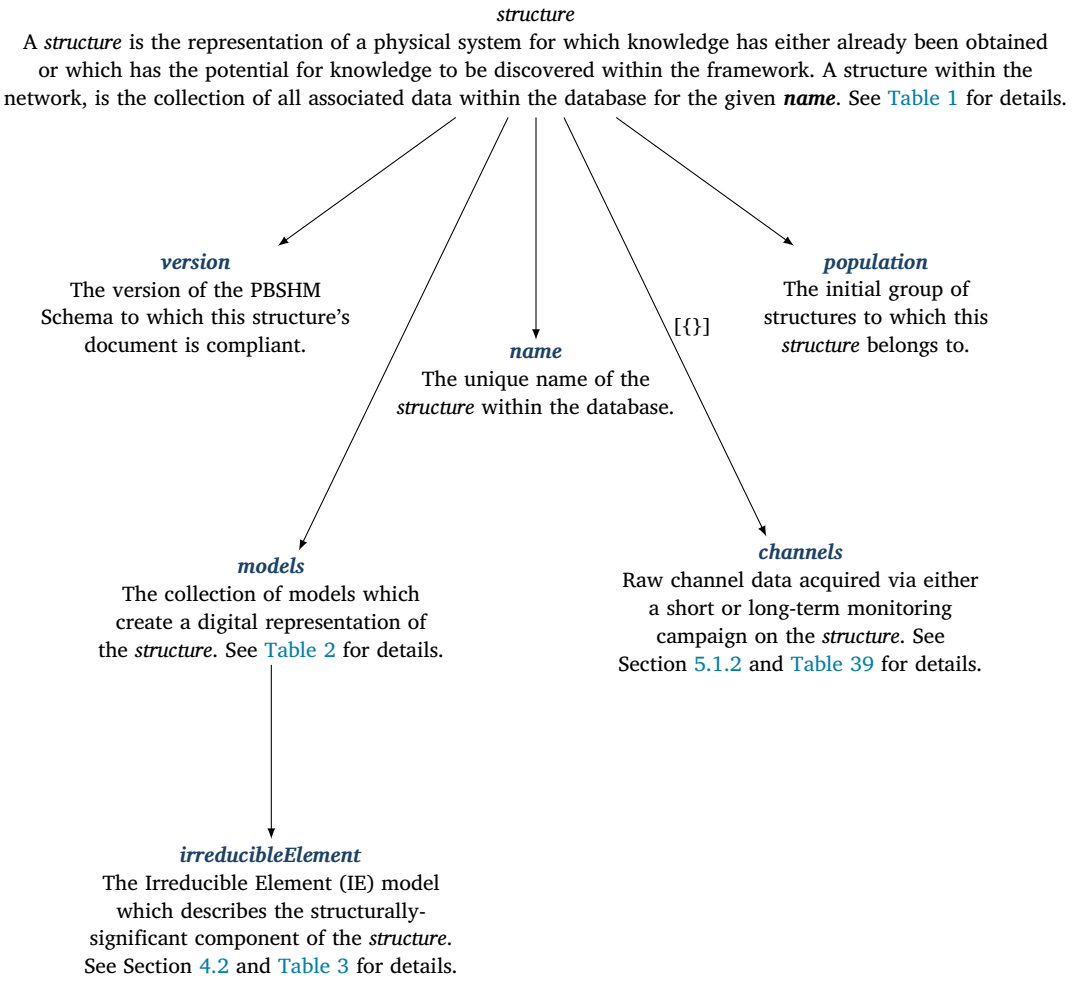


Fig. 22. The hierarchical layout of properties and child objects within the *structure* root document.

5.2. 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 paper and the JSON code required to represent data into the database. Fig. 24 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.

Fig. 25 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 paper in the interest of brevity, a single JSON document could contain both IE model data and channel data.

5.3. Framework

In the interest of conciseness, the complete implementation of the framework is omitted from this paper; 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 3.2, 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.

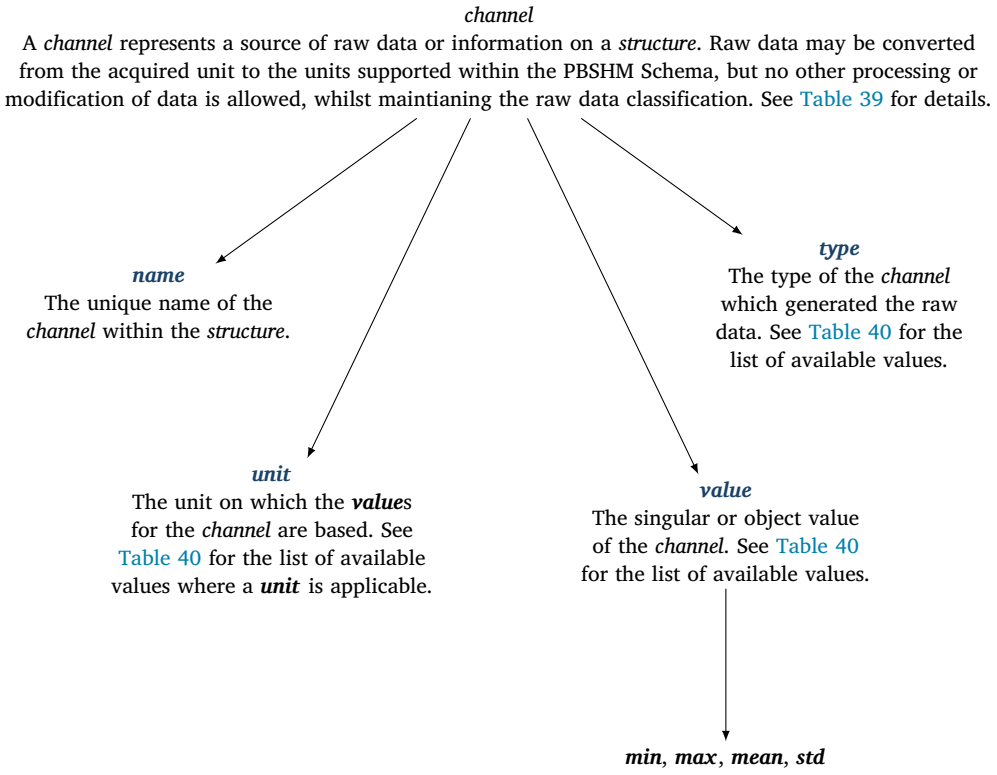


Fig. 23. The properties for a *channel* in the PBSHM Schema.

Keeping in mind the requirements outlined within this paper of what is required for a PBSHM framework, two modules have initially been authored to fulfil the aforementioned requirements. Fig. 29 in the Appendix shows a module within the PBSHM framework which is an implementation of the Jaccard Index originally used within Gosliga et al. [20] paper to determine the similarity between a set of real-world bridges. The IE models of the associated bridges, have been updated to the latest version of the IE model definitions included within this paper. Fig. 30 in the Appendix shows the 'pathfinder' module within the framework which enables exploration of existing datasets within the database and includes basic statistical analysis of these datasets.

5.4. Download

The PBSHM Schema and PBSHM Framework introduced in this paper 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 PBSHM Schema repository is *structure-data-compiled-mongodb.min.json*. To configure the PBSHM Framework, please follow the instructions in the *README* file on the PBSHM Framework repository. During the configuration of the PBSHM Framework, the latest version of the PBSHM Schema will be downloaded and installed within the selected *collection*.

6. Conclusion

In conclusion, this paper has highlighted some significant data problems that could potentially plague PBSHM, if not dealt with whilst PBSHM is in its foundation stages. The paper has proposed the potential solution for this, 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 PBSHM henceforth to be on algorithmic discovery and generation, instead of data headaches.

This paper 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 relationships. The *framework* is where PBSHM algorithms reside; any computations on the similarity of structures or determining potential transfer of knowledge from structures, happen here.

```
{
    "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" }
                    ]
                }
            ]
        }
    }
}
```


Fig. 24. 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*.

The *database* unifies both the *network* and *framework* by providing the shared data format against which all PBSHM data must be compliant.

The paper also takes the initial concept of Irreducible Element (IE) models introduced by Gosliga et al. [3] and proposes a new and expanded version – and subsequent language – of IE models, where attributes have a clear definition as to their purpose within the model and what information they should contain. The improved language further enables embedding of additional engineering knowledge within the model and encapsulates certain design choices made by the modeller.

The importance of the changes outlined within this paper to the language and knowledge embedding within an IE model become apparent when one examines the output from the framework's similarity metrics for the eight real-world bridges used by Gosliga et al. [20]. Fig. 26(a) 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. Fig. 26(b) shows the output for from the framework for the same eight bridges, but instead embedding the contextual type within the Attributed Graph. As one can

```
{
    "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"
        }
    ]
}
```


Fig. 25. 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.

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

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

Fig. 26. 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 real-world bridges used by Gosliga et al. [20].

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 within this paper to isolate embedded knowledge to their corresponding unique domains.

6.1. Future research

Whilst the work outlined within this paper has proposed an improved standardisation for Irreducible Element (IE) models, there is still further work to be concluded on the topic of embedding knowledge on a structure within PBSHM. An IE model is currently the only vehicle within PBSHM for embedding the knowledge regarding the composition of a structure; however, there are vast amounts of additional data, such as Environment and Operational Variables (EOV's), which are important indicators of the overall state of a structure, whilst not aligning to the purpose of an IE model. It is not envisioned that the purpose of an IE model will change to encapsulate this data; instead, a new model will be birthed: the *Reality* model.

The *Reality* model will be a hierarchical model which is dependent upon an IE model to describe the structure in question, with additional models/data added on top of the IE model to gain additional context regarding the structure's interactions within the world in which the structure lives. Part of the additional information that is required to be present in the hierarchical *Reality* model, is information regarding the network of sensors placed upon the structure. Being able to locate a sensor or sensors within a [regular] *element* is key to gaining a detailed understanding of general Structural Health Monitoring (SHM) problems within PBSHM.

There is also further research within the IE model itself which needs to be conducted. There is currently no method within an IE model for the modeller to inform the *framework* and subsequently the *network* regarding the granularity of the model. Have they 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? This information is desired to be present within the model, so to inform any similarity algorithms during matching. There may also be the need for different types of IE models with differing levels of details on *element*s and *relationship*s because of the purpose they may have during the lifecycle of the PBSHM ecosystem.

Currently, within an IE model, there is only a reference to an external structure/object being present; [ground] *element*. It is envisioned that this methodology will need to be expanded in future versions of the IE model language, enabling referencing of other IE models within the *network* as to gain additional context within the referencing IE model. This principle may also be expanded to encompass the referencing of subsections of IE models. If one imagines an aeroplane, 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 all the intricate *element*s and *relationship*s which form the landing gear section, they should be able to declare a new *element* which simply references the relevant external – to itself – IE model.

The referencing of external IE model methodology, may also be expanded to encapsulate named generic subsections or subcomponents within a model. If one again imagines an aeroplane, the purpose of the IE model generation could be to determine similarities within cockpit designs across a range of aircraft. In this scenario, it would be preferential for the modeller, to be able to reference a generic 'wing' to include within the IE model, as the specifics regarding the wing design is not required for the purpose in which the IE model is being generated and used within the *network*.

CRediT authorship contribution statement

Daniel S. Brennan: Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Julian Gosliga:** Conceptualization, Methodology, Software. **Elizabeth J. Cross:** Supervision, Writing – review & editing. **Keith Worden:** Conceptualization, Funding acquisition, Resources, Supervision, Writing – review & editing.

Data availability

The links to download the code used are included in the download section of this paper.

Acknowledgements

The authors of this paper gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council (EPSRC) via grant reference EP/W005816/1. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising. The authors of this paper also gratefully acknowledge Andrew Bunce and David Hester of Queen's University of Belfast for providing the photographs of the example bridge and associated IE models used within the similarity metrics. The authors of this paper gratefully acknowledge Connor O'Higgins of Queen's University of Belfast for providing the channel data used for generating the framework's channel screenshots. The authors also gratefully acknowledge Chandula T Wickramarachchi for her initial thoughts on the requirements of a PBSHM framework.

Appendix

This appendix necessarily has something of a 'user manual' feel to it, and might be considered to be outwith the remit of a research paper proper; however, it is included here because the detailed structure of some of the objects referred to in the main text shed valuable light on their discussions there.

Appendix A. Root schema

See Tables 1 and 2.

Table 1
The available properties for the declaration of a structure within the PBSHM 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 2 for the available child properties.	*object*
channels	Associated raw channel data acquired from either a short or long-term monitoring campaign. See Table 39 for the available child properties and see Table 40 for a list of accepted values.	

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

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

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

Required Properties: None.

Appendix B. IE model schema

See Table 3.

Table 3
The available properties for the declaration of an Irreducible Element model within the PBSHM 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 Fig. 9 for the available *element*s dependent upon the selected *model* type.	*array*
relationships	An array of *relationship* objects. See Fig. 9 for the available *relationship*s dependent upon the selected *model* type.	*array*

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

B.1. Regular element

See Table 4.

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

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 5 for the available child properties.	*object*
contextual	Purpose of the [regular] *element* has within the *model*. See Table 7 for the available child properties.	*object*
geometry	Details of the shape and volume that the [regular] *element* occupies within the *model*s coordinate space. See Table 8 for the available child properties.	*object*
material	Details of the material and associated properties that the [regular] *element* is constructed from. See Table 15 for the available child properties.	*object*

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

B.1.1. Coordinates

See Tables 5 and 6.

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

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

Required Properties: ***global***.

Table 6
The available properties for the global 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 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 35 for the available child properties.	*object*

Required Properties: ***translational***.

B.1.2. Contextual

See Table 7.

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

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***.

B.1.3. Geometry

See Tables 8–14.

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

Property	Description	Type
type	The shape that describes the volume which the *element* occupies. See Table 34 for the available nested child properties. See Table 9 for the list of accepted standard types and Table 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 10 for a list of child properties.	*object*
faces	The dimensions and properties of each face of the complex shape being described. See Table 11 for a list of child properties. Only available when the shape is a complex geometry (See Table 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 9 and 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 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 9 for standard geometrical types and in Table 14 for complex geometrical types.	*object*

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

Table 9
The available standard geometrical types and their associated required dimensions properties and accepted object types for 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 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 38 for the available child properties and, within the table, 'Wildcard' under 'Dimension Object Type' for the accepted values..

Table 10
The available properties for the cuboid bounding box object within 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 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 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 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 11
The available properties for the complex geometrical faces 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 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 12 for a full list of child properties.	*object*

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

Table 12
The available properties for the complex geometrical face side 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 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 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 14.	*object*
translational	Y and Z translational values within the bounding box coordinate space to convey the translational movements of the face. See Table 13 for a full list of child properties.	*object*

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

Table 13
The available properties for the complex geometrical face translation 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 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 36 for the available child properties and, within the table, 'Linear' under 'Value Object Type' for the accepted values.	

Required Properties: ***y*** and ***z***.
Additional Information: The values for the translational movements must be relative to the bounding box dimensions.

Table 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.

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 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 38 for the available child properties and, within the table, 'Linear' under 'Dimension Object Type' for the accepted values.

B.1.4. Material

See Tables 15–21 and Fig. 27.

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

Property	Description	Type
type	The classification of the material that composes the [regular] *element*. See Fig. 27 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 16	*array*

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

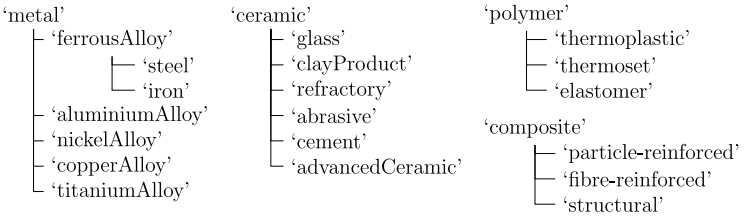


Fig. 27. The available material type tree for a [regular] *element* in the PBSHM schema.

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

Property	Description	Type
type	The type of property on the material within the [regular] *element*. See Tables 17 and 21 for accepted values which do not support a ***unit*** value and see Table 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 19 for a list of accepted child properties for a conditional value. Singular values are supported by types in Tables 17 and 18 and conditional values are supported by types in Tables 17, 18 and 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 18.

Table 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
'PoissonsRatio'
'elongation'
'reductionInArea'
'fatigueStrengthExponent'
'fatigueDuctilityCoefficient'
'fatigueDuctilityExponent'

Table 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.

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

Property	Description	Type
environmental	The test environmental conditions associated with the given value. See Table 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 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 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 21 which the accepted values are under 'Value Object Type' in Table 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 20
The available properties for the environmental 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 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 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 36 for the available child properties and, within the table, 'Wildcard' under 'Value Object Type' for the accepted values..

Table 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.

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

B.2. Ground element

See Table 22.

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

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***.

B.3. Perfect & Boundary relationship

See Tables 23–26.

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

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 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 25 for the available child properties.	*object*

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

Table 24
The available properties for named element object within a [perfect] and [boundary] *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*

Required Properties: ***name***.

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

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

Required Properties: ***global***.

Table 26
The available properties for the global 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 35 for the available child properties.	*object*

Required Properties: ***translational***.

B.4. Connection relationship

See Tables 27 and 28.

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

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 28 for the available child properties.	*array*

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

Table 28
The available properties for named element object within a [connection] *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*
nature	The nature of physics that would construct the joint. See Table 30 for a list of child properties and see the values under the 'static' root of the tree in Fig. 28 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 25 for the available child properties.	*object*

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

B.5. Joint relationship

See Tables 29–33 and Fig. 28.

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

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 30 for a list of child properties and see Fig. 28 for a tree of accepted values.	*object*
degreesOfFreedom	The allowed movement within the joint. See Table 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 33 for the available child properties.	*array*

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

Table 30
The available properties for a nested nature object within a [joint] or [connection] *relationship* in 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 30 for the accepted child properties.	*string*

Required Properties: ***name***.

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

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

Required Properties: ***global***.

Table 32
The available properties for the global 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 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 35 for the available child properties.	*object*

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

Table 33
The available properties for named element object within a [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 *models* coordinate space. See Table 25 for the available child properties.	*object*

Required Properties: ***name***.

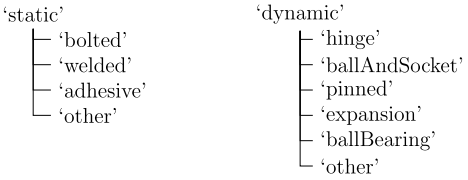


Fig. 28. The available nature type tree within a [joint] or [connection] *relationship* in the PBSHM schema. When declaring the nature within a [connection] *relationship*, only the 'static' portion of the nature tree is available.

B.6. Shared objects

See Tables 34–38.

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

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 34 for the accepted child properties.	*object*

Required Properties: ***name***.

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

Translational coordinates object		
Property	Description	Type
x, ***y***, ***z***	The translational values within the coordinate space. See Table 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 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 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 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 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 ***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 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
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 38
The available properties for a dimension object and associated accepted values for the different types of dimension 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

Appendix C. Channel schema

See Tables 39–41.

Table 39
List of channel object properties in the PBSHM 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 40 for the list of available options.	*string*
unit	The selected unit value of this channel on which the value is based; see Table 40 for the list of available options for the selected type.	*string*
value	Value of the channel, stored in the selected unit.	*See Table 40*

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

Table 40
The available values for Channel properties within 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 41
List of value object properties in 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.

Appendix D. System outputs

See Figs. 29 and 30.

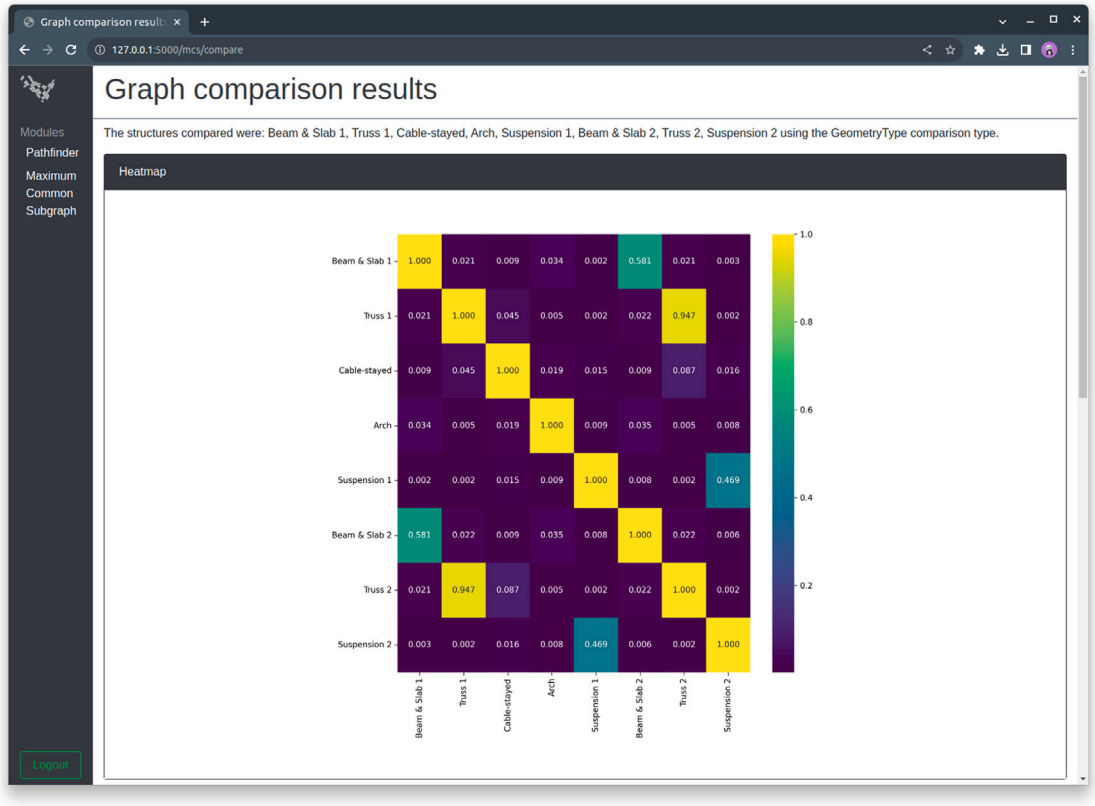


Fig. 29. An output from the PBSHM framework on computing the similarity of eight real-world bridge structures using the Jaccard Index. The bridges included within the comparison are the same set of bridges used by Gosliga et al. [20].

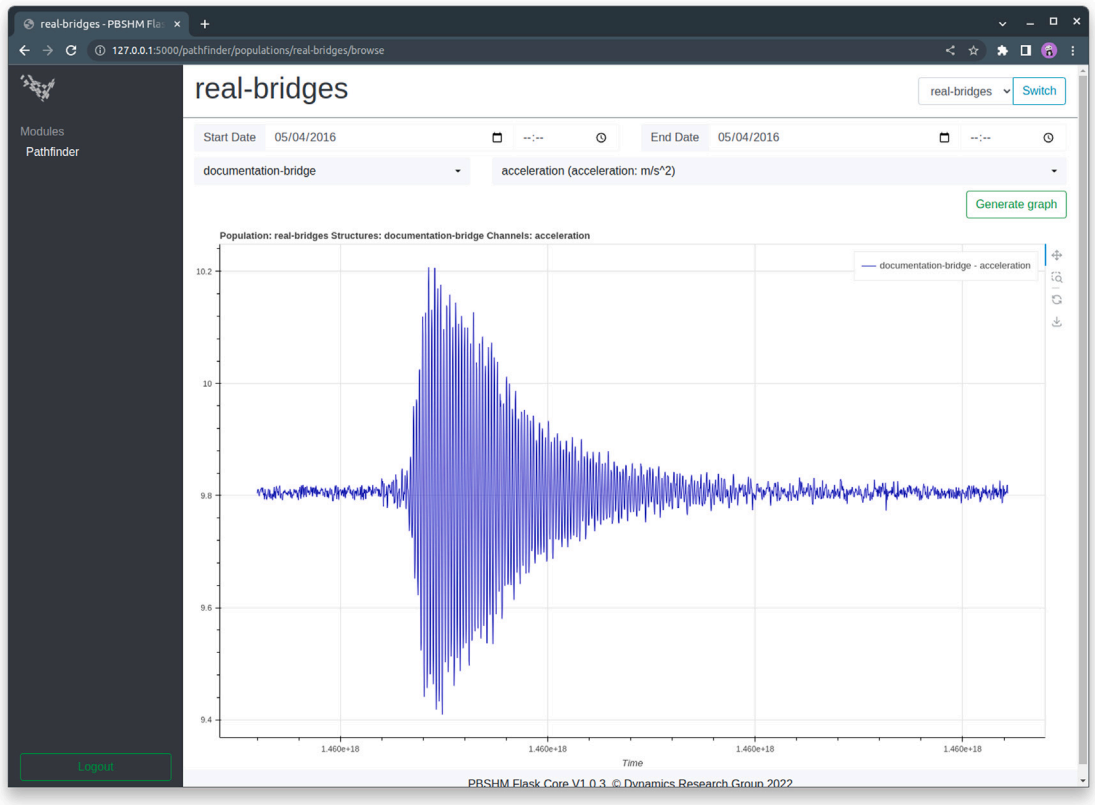


Fig. 30. An output from the PBSHM framework on browsing the available *channel* data for a given population.

References

[1] C.R. Farrar, K. Worden, An introduction to structural health monitoring, Phil. Trans. R. Soc. A 365 (1851) (2007) 303–315.
[2] L.A. Bull, P.A. Gardner, J. Gosliga, T.J. Rogers, N. Dervilis, E.J. Cross, E. Papatheou, A.E. Maguire, C. Campos, K. Worden, Foundations of population-based SHM, Part I: Homogeneous populations and forms, Mech. Syst. Signal Process. 148 (2021) 107141.
[3] J. Gosliga, P.A. Gardner, L.A. Bull, N. Dervilis, K. Worden, Foundations of population-based SHM, Part II: Heterogeneous populations – Graphs, networks, and communities, Mech. Syst. Signal Process. 148 (2021) 107144.
[4] P. Gardner, L.A. Bull, J. Gosliga, N. Dervilis, K. Worden, Foundations of population-based SHM, Part III: Heterogeneous populations – Mapping and transfer, Mech. Syst. Signal Process. 149 (2021) 107142.
[5] G Tsialiamanis, C. Mylonas, E. Chatzi, N Dervilis, D.J. Wagg, K Worden, Foundations of population-based SHM, Part IV: Structures and feature spaces as geometry, Mech. Syst. Signal Process. 157 (2021) 107692.
[6] C.R. Farrar, K. Worden, Structural Health Monitoring: A Machine Learning Perspective, Wiley, Chichester, West Sussex, U.K. ; Hoboken, N.J, 2013.
[7] K. Worden, L.A. Bull, P. Gardner, J. Gosliga, T.J. Rogers, E.J. Cross, E. Papatheou, W. Lin, N. Dervilis, A brief introduction to recent developments in population-based structural health monitoring, Front. Built Environ. 6 (2020) 146.
[8] K. Worden, E.J. Cross, N. Dervilis, E. Papatheou, I. Antoniadou, Structural health monitoring: From structures to systems-of-systems, IFAC-PapersOnLine 48 (21) (2015) 1–17.
[9] S. Jeong, J. Byun, D. Kim, H. Sohn, I.H. Bae, K.H. Law, A data management infrastructure for bridge monitoring, in: Jerome P. Lynch (Ed.), SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring, San Diego, California, United States, 2015, p. 94350P.
[10] E.F. Codd, A relational model of data for large shared data banks, Commun. ACM (1970) 11.
[11] C. Nance, T. Losser, R. Iype, G. Harmon, NOSQL vs RDBMS - Why there is room for both, 2013, p. 7.
[12] R. Zafar, E. Yafi, M.F. Zuhairi, H. Dao, Big data: The NoSQL and RDBMS review, in: 2016 International Conference on Information and Communication Technology, ICICTM, IEEE, Kuala Lumpur, Malaysia, 2016, pp. 120–126.
[13] A. Gandini, M. Gribaudo, W.J. Knottenbelt, R. Osman, P. Piazzolla, Performance Evaluation of NoSQL Databases. p. 15.
[14] S. Bradshaw, E. Brazil, K. Chodorow, MongoDB: The Definitive Guide, third ed., O'Reilly, 2019.
[15] MongoDB, Inc, Encryption at rest - MongoDB manual, 2020, https://docs.mongodb.com/manual/core/security-encryption-at-rest/.
[16] MongoDB, Inc, Users - MongoDB manual, 2020, https://docs.mongodb.com/manual/core/security-users/\#sharded-cluster-users.
[17] MongoDB, Inc, TLS/SSL (transport encryption) - MongoDB manual, 2020, https://docs.mongodb.com/manual/core/security-transport-encryption/.
[18] E.F. Codd, Further Normalization of the Data Base Relational Model, in: Courant Computer Science Symposia 6, Data Base Systems, Prentice-Hall, 1972.
[19] D. Crawford, Technical Correspondence. p. 2.
[20] J. Gosliga, D. Hester, K. Worden, A. Bunce, On population-based structural health monitoring for bridges, Mech. Syst. Signal Process. 173 (2022) 108919.
[21] D.S. Brennan, J. Gosliga, P.l Gardner, R.S. Mills, K. Worden, On the application of population-based structural health monitoring in aerospace engineering, Front. Robot. AI 9 (2022) 840058.

[22] D.S. Brennan, T.J. Rogers, E.J. Cross, K. Worden, On calculating structural similarity metrics in population- based structural health monitoring, Data Centric Eng. (2024) submitted for publication.
[23] MongoDB, Inc, About us - MongoDB, 2020, https://www.mongodb.com/company.
[24] T. Bray, RFC 7159 - The JavaScript object notation (JSON) data interchange format, 2020, https://tools.ietf.org/pdf/rfc7159.pdf.
[25] T. Bray, RFC 8259 - The JavaScript object notation (JSON) data interchange format, 2020, https://tools.ietf.org/pdf/rfc8259.pdf.
[26] MongoDB, Inc, Role-based access control - MongoDB manual, 2020, https://docs.mongodb.com/manual/core/authorization/.
[27] MongoDB, Inc, Storage engines - MongoDB manual, 2020, https://docs.mongodb.com/manual/core/storage-engines/.
[28] MongoDB, Inc, WiredTiger storage engine - MongoDB manual, 2020, https://docs.mongodb.com/manual/core/wiredtiger/\#document-level-concurrency.
[29] MongoDB, Inc, Indexes - MongoDB manual, 2020, https://docs.mongodb.com/manual/indexes/index.html.
[30] MongoDB, Inc, Replication - MongoDB manual, 2020, https://docs.mongodb.com/manual/replication/.
[31] MongoDB, Inc, Sharding - MongoDB manual, 2020, https://docs.mongodb.com/manual/sharding/index.html.