Journal of Machine Engineering, Vol. 16, No. 4, 2016

Received: *11 November 2016* / Accepted: *01 July 2016* / Published online: *09 December 2016*

machine design, automated design systems, structural feature language, antipatterns

Wojciech KACALAK[1]
Maciej MAJEWSKI[1]
Andrzej TUCHOLKA[1*]

A METHOD OF OBJECT-ORIENTED SYMBOLICAL DESCRIPTION AND EVALUATION OF MACHINE ELEMENTS USING ANTIPATTERNS

Authors propose a method of normalizing and analyzing structures, enabling automated cross-comparison of features observed in tested structures against predefined ones (both correct and incorrect reference elements). Application of this method, enables usage of numerical methods and neural networks for rapid classification of elements, while maintaining high level of flexibility, uncertainty and variety of input data. Additionally, a language, enabling symbolic representation of mechanical elements is presented and used in the demonstration of the proposed method applied to mechanical shafts. Authors explore one simple, and two more complex calculation models aimed at automation of the evaluation and rating of mechanical elements, with a purpose of improving the design and quality monitoring processes. Increase in efficiency of these processes results from applying proposed method as a tool for reducing common mistakes, faults, quality assessment time of large amounts of similar elements, or automatic suggesting of possible solutions.

1. INTRODUCTION

One of the key challenges in the design of mechanical structures and structural analysis, is the ability of engineers to meaningfully evaluate their structural properties and character [11],[14]. The quality and speed with which such evaluation can be performed, is a key element of the design, construction, diagnostic, and maintenance processes.

Existing methods require manual creation of models for each of the tested structures, or executing simulations which (due to their nature) either lack in speed or precision when evaluating complex models. Both of these limitations make it very difficult to apply genetic or unsupervised machine learning algorithms, where potentially vast scope of solutions has to be quickly evaluated.

Additional difficulty lies in automated interpretation of the reference data, and defining such information in a form of a rich and quantitative description of the tested structure. Inclusion of such knowledge provides an opportunity for a rapid and cumulative increase in quality of designed structures (due to ease of exchange of data), and also enables advanced solution-finding algorithms to enhance the collaborative intelligence of human

[1] Koszalin University of Technology, Faculty of Mechanical Engineering, Koszalin, Poland
[*] Email: andrzej.tucholka@gmail.com

operators and their tools. Furthermore, the particular use of antipatterns - antipatterns in this context are considered as known and incorrect examples of structures. [1],[8],[9],[12] provides the algorithms with the ability to explore an open set of possible correct solutions, taking as the only reference already identified, incorrect ones.

Proposed method enables cross-testing of structures providing mechanical designers with an ability to check different options against each other, but also easily including known, predefined solutions in the analysis (be it correct ones or antipatterns).

2. OBJECT-ORIENTED CHARACTER OF THE FEATURE DESCRIPTION

A part of the proposed system is our symbolical language used for defining structural properties of selected machine components during the design stage. Its purpose is to enable creation of descriptions of fundamental objects, such as elements of a structure.

In order to ensure unambiguous description and correct semantics of the language, a method of distinguishing individual elements constituting machine parts has been defined. It has been specified that each structural object has its own attributes (e. g. length, diameter, etc.) and is comprised of constituent objects. As an example, for a structural object being a machine shaft, constituent objects are distinguished based on their position, dimensions and shape, as well as their surface layer properties. Constituent objects are simple polyhedra and they have their own attributes: dimensions, shape and position. In the given example a step of the shaft is a constituent object.

Any changes of dimensions, shape and surface layer properties are made with modifiers. A modifier also has its own attributes, which are defined in relation to the modified constituent object (parent object), to another constituent object, or to another modifier. A modifier cannot exist independently. Therefore, a modifier is an integral part of its parent object. Examples of a modifier are chamfer, a hole, or a thread.

The main assumption of the symbolical notation of structural features of machine parts' elements is that their description is object-oriented. Objects are characterized by attributes and permitted operations. Attributes are properties or variables associated with the object, and operations are functions or operations performed on object's attributes for the purpose of auto-modification or in order to influence other objects. Access to an object and its attributes is possible only through a defined interface. The interface interacts with an object's instance – which can be perceived as a living specimen (instance) of a species (object). Functionality of an object is directly connected with the attributes associated to it.

A disadvantage of the – currently used – structural description, usually in the form of a tree, less often graphs, is lack of description of relations between objects. When it comes to creation of technological and organizational processes, object-based description is important due to the manner of data notation.

Object-oriented notation for structural features requires creation of classes representing sets of objects, as well as methods (functions) allowing to perform an array of operations on particular objects, such as creation, deletion, modification, moving, rotation, etc. Objects of one class share many similarities, but they can insignificantly differ

from each other. A class defines which operations and attributes are associated with an object, but values assigned to attributes may vary between instances.

Complex structures can be described in many ways. It can be done using an XML or JSON compatible notations, a specialized declarative language similar to HTML, or in a minimized or symbolical notation. We have defined assumptions for creation of a language, whereby a structural object can be easily divided into constituent objects, and modifiers can be distinguished as elements for altering dimensions, shape and properties of constituent objects. Their symbolical minimized notation has been used in this is article to demonstrate the language in a simple and easy to read manner.

Symbolical language is a form of machine notation used by a system, which generates, and interprets such a notation for automatic real-time creation of designed components. For the convenience of designing engineers, a higher-level language can be used, such as a constructional-XML.

3. RELATIONS BETWEEN OBJECTS' GEOMETRICAL PROPERTIES

In automated systems for designing machine components and assemblies [2],[3],[4],[5],[6],[7], relations between an object and element's geometrical properties, as well as geometrical relations between those objects are key. A diagram of such relations, by the example of a single key-type sleeve coupling, is presented in the Fig. 1. The geometrical properties identification subsystem consists of the following types of geometrical relations:

- 1st relation type - is present in a particular element of a given type class. In the presented example, it is a dimensional relation determining the position of the axis of the hole in the crankpin (element 1) in relation to its facing surface. And for the object of class sleeve it is a dimension marking the distance between the hole's axis (element 1), and its facing surface;

- 2nd relation type - determines the geometrical relation between geometrical properties of different elements of a given object (machine component). For a shaft and a sleeve of a sleeve coupling these are dimensions determining the distance between holes' axes. The axis of the first hole is connected with the crankpin (element 1), and the axis of the other hole belongs to the shaft's step (element 2). The situation is analogous in the case of the sleeve object;

- 3rd relation type - geometrical relations between different objects in an assembly. In machine designs these are relations between geometrical properties of the 1st and 2nd type, belonging the object of a given class having such relations with other classes of objects. Third-type relations usually specify mutual position of construction's components. In a single key-type sleeve coupling it is crucial to ensure accuracy of mutual position of the shaft hole's axis (Element 1) and the sleeve hole's axis (Element 1). For this reason the third-type relation, in the context of the provided example, means that the axes of the aligned holes must be coaxial.

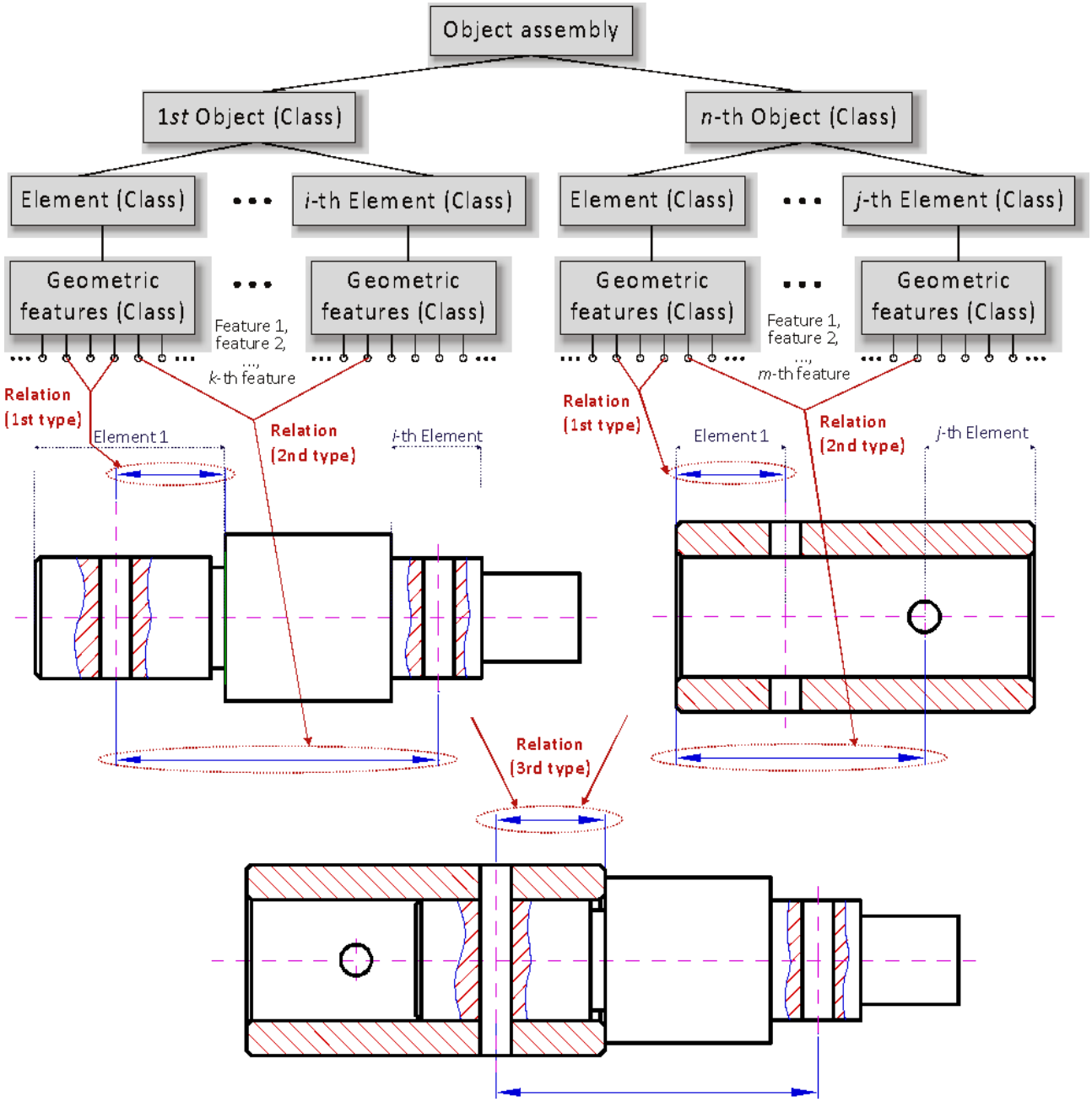


Fig. 1. Relations between an object, of a given class, element's geometrical properties and geometrical relations between those objects by the example of a single key-type sleeve coupling

4. KXML REPRESENTATION OF STRUCTURES

Assuming a reference classification of structures i.e. ISO (International Standards Organization) or PKN (Polish Normalization Committee) grants us with the ability to assign relevant features and calculation methods to the model. It is obvious that the features of pipes can be reused among them, while being different from ones found in electric wires or mechanical joints. Still, for most purposes, one can use any of commonly used

classification models to satisfy the need for a general abstraction and interpretation of structures within a specific context (defined by the reference classification method).

Existing structural data formats are often designed mainly for manufacturing design and data visualization purposes. Both of these use-cases promote precise and complete definition of data, rather than correct abstraction and adequateness of its structure for a specific calculation model. Still, provided adequate interpretation and parsing mechanisms, proposed normalization method can be used with different structures of the input data, while the breadth and depth of the existing data exchange standards (i.e. ISO 10303) makes it easier to classify and interpret the structural data of the mechanical elements.

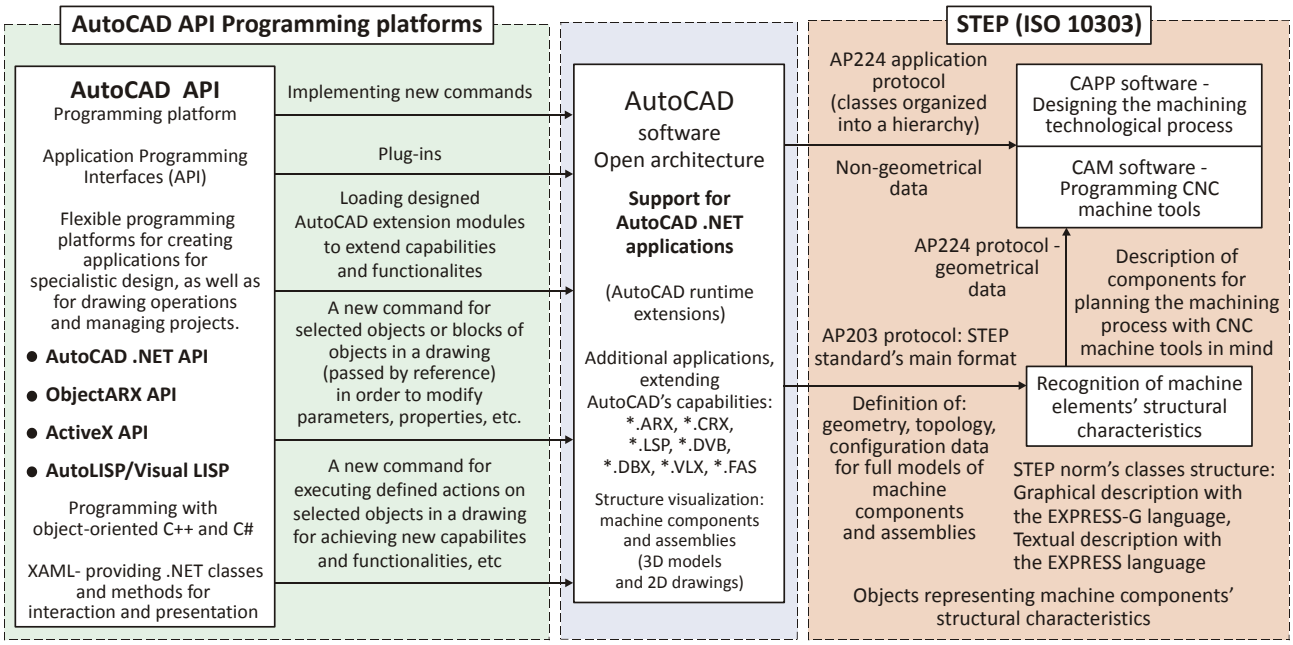


Fig. 2. Integration interfaces overview for AutoCAD and ISO 10303

Proposed data normalization and serialization processes reduce the structural character of the chosen notation, yet it is important to choose the container notation or format (i.e. XML) able to carry information (varying in completeness, structuring, naming, versioning, formatting) for practical usage in specific numerical methods and related data processing functions. The proposed representation of the structure is built upon a set of models, defining a set of features used in it to describe its tree of nodes defining the structural dependencies.

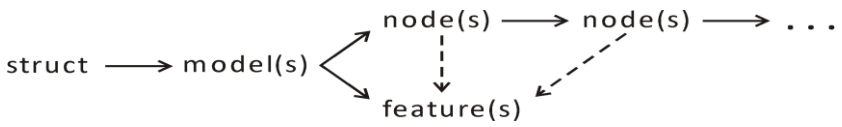
To enable the possibility of using a library of predefined structures, this syntax can be used in two manners: as a pre-defined template of a structure, or a description of a concrete structure. The difference between these can be observed after verifying if all of the features required for a given class of a node in the particular model, have been provided with values. Only if all information (required to validate the model as complete) has been provided, the model can be considered a concrete structure.

Table 1. Key KXML structures used to define the structure

Plain text	Object-oriented and symbolic representation (KXML)
define a shaft named s001	<**struct class**="shaft" **id**="s001"> </**struct**>
using the mechanical model	<**model class**="mechanical"> </**model**>
define measured features as: l in in millimeters, R in millimeters, d in millimeters, and angle in degrees on a vertical 3d axis.	<**feature id**="l" **unit**="mm" /> <**feature id**="R" **unit**="mm" /> <**feature id**="d" **unit**="mm" /> <**feature id**="angle" **unit**="deg" **vector**="(0,1,0)" />
define a spigot named p001, with l equal to 48, and d equal to 35, including a mill with l equal to 12, and angle equal to 45, a thread with l equal to 24, and an undercut with R and d equal to 8	<**node class**="spigot" **id**="p001"> <**feature id**="l">48</**feature**> <**feature id**="d">35</**feature**> <**node class**="mill"> <**feature id**="l">12</**feature**> <**feature id**="angle">35</**feature**> </**node**> <**node class**="thread"> <**feature id**="l">24</**feature**> </**node**> <**node class**="undercut"> <**feature id**="R">8</**feature**> <**feature id**="d">8</**feature**> </**node**> </**node**>

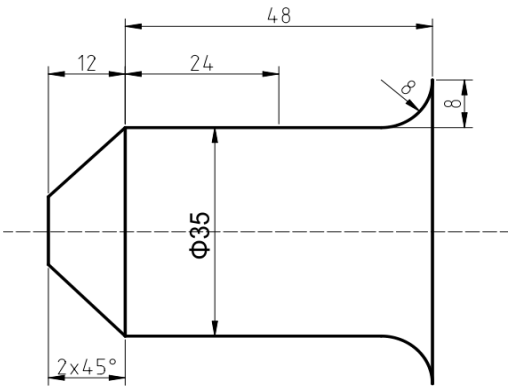
The additional flexibility built into this method is based on the optional support of both: composition and inheritance between structural nodes via the class attribute. Both: the class based relations and the library mechanism, provide a common denominator that can be used for batch processing or flexibly structuring the analysis.

Table 2. Presentation of the syntax's structure



<**struct class**="shaft" **id**="s001">
<**model class**="mechanical">
<**feature id**="l" **unit**="mm" />
<**feature id**="R" **unit**="mm" />
<**feature id**="d" **unit**="mm" />
<**feature id**="angle" **unit**="deg" **vector**="(0,1,0)" />
<**node class**="spigot" **id**="p001">
<**feature id**="l">48</**feature**>
<**feature id**="d">35</**feature**>
<**node class**="mill">

```
    <feature id="l">12</feature>
    <feature id="angle">45</feature>
   </node>
   <node class="thread">
    <feature id="l">24</feature>
   </node>
   <node class="undercut">
    <feature id="R">8</feature>
    <feature id="d">8</feature>
   </node>
  </node>
</model>
</struct>
```



Each of the keywords used in this notation provides both: a structural and a functional interpretation (dependant on the class and other attributes):

- STRUCT - a root KXML element providing a top level, singular access to the contained structure; storing multiple struct nodes in one KXML file is allowed and supported for easy and flexible creation of collections of structures;
    - class - an optional attribute relating the structure with a specific and uniquely identified class of elements;
    - id - an optional attribute specifying the symbolic name of this structure; to avoid confusion it is advised to keep the ids of the structures unique within at least the KXML file and optionally define namespace based on the file system;
- MODEL - an element of the syntax used to define a model used for analysis while defining the trees of NODEs contained inside.
    - class - a required attribute allowing the choice of an appropriate model for interpretation of the data contained in this element; the values provided in the class attribute are expected to be unique within the scope of one structure;
    - id - an optional attribute enabling the referencing of the structure's nodes or features in a view limited only to a single model in contrast to the default way of looking at a complete structure;
- FEATURE - an element on one side defining a reused (between KXML elements), measurable property of the structure, model or a node, and also providing concrete values describing the structure;
    - id - a required attribute enabling identification of the structural feature for the purpose of de-referencing it, or templating via the class field (both described below);
    - class - an optional attribute enabling relating (through inheritance) the features with each other; this can be used for abstracting common features used in the structure, but also transmits information about the implicit relations between features;
    - unit - an attribute of the feature element, that is required to be defined at least in one of the feature elements with the same id;

  - vector - an optional attribute used for features of the structure that have a vector assigned to them;
  - value - an optional value of the KXML containing a concrete number (or a math function returning one) representing the value of the feature measured in defined units;
- NODE - the main structural element of KXML allowing the definition of the structure of the description through nesting of the elements, ability to differentiate between the types of the relations between nodes for each of the models, and to relate the nodes with each other to represent implicit relations between the nodes;
  - id - an optional attribute enabling referencing of the node in the feature value, or for templating purposes via the class field;
  - class - a required attribute defining an interpretation and the relations between the nodes.

The ability to include the feature or a given structure of nodes in the analysis comes from its identification with a formal scientific method. Normally the scope of features will directly depend on the chosen classification model, but it is important to note, that the feature scopes might overlap among the chosen models.

4.1 RELATIONS BETWEEN ELEMENTS OF THE SYMBOLIC LANGUAGE

The structures in the proposed language, are defined using differently classified models. Such classification, for the purpose of the symbolic language, is considered as a reference for: definition of the names, approach to number formatting, scope along with the detail of provided information, and abstraction of what can be considered a node. This approach also ensures, that the data which has been provided for processing purposes, has a concise and concrete meaning to avoid unnecessary noise in its numerical analysis.

The result symbolical structure (RDCK Tree) is derived from these main principles of combining the keywords of the language:

- structure of the elements of the language;
- correlations of class definitions of all elements of the language;
- inheritance and templating the elements of the language;
- identity sharing among nodes of different models.

For the purposes of achieving a more complete description of relations between the nodes, it is possible to reference nodes not only within one model, but also between them. Notice, that as much as the features of the nodes are comparable only within the same model (or referencing the same feature definition in the same model classification), the referencing relation between the nodes from separate models should be only structural (this is node the same node as...) or optionally value references (this value of the feature is calculated based on the value of ...).

The structure of nodes created for each of the models, allows to place only one node in the tree structure built within the model. The node ids have to be unique in the model or be automatically assigned a unique id in case no identifier has been provided. As mentioned above, reusing of the node id between models is interpreted as co-identity. While nodes

contained within the model form trees linked at least on the root level of nodes, and as described in this chapter, linking the nodes and features in a specific manner.

Table 3. Four model based relation classes with examples

<table>
<tr><td colspan="2">Sharing the common definition of the feature by the structure of nodes in the same model</td></tr>
<tr><td><model class="mechanical">
<feature id="l" unit="mm" />
<feature id="R" unit="mm" />
<feature id="d" unit="mm" />
<node class="spigot" id="p001">
<feature id="l">48</feature>
<feature id="d">35</feature>
<node class="mill">
<feature id="l">12</feature>
</node>
<node class="thread">
<feature id="l">24</feature>
</node>
<node class="undercut">
<feature id="R">8</feature>
<feature id="d">8</feature>
</node>
</node>
</model></td><td>Each feature instance, regardless of the level of nesting the node, is using the same feature definition provided in the model.

Example on the left includes the feature “l” - the length measured in millimeters, and being provided a concrete value of the spigot, mill and the thread nodes.</td></tr>
<tr><td colspan="2">Sharing the information about node’s identity between different models</td></tr>
<tr><td><model class="mechanical">
<feature id="l" unit="mm" />
<feature id="d" unit="mm" />
<node class="spigot" id="p001">
<feature id="l">48</feature>
<feature id="d">35</feature>
</node>
</model>
<model class="electrical">
<feature id="rho" unit="Ohm*m" />
<node class="conductor" id="p001">
<feature id="rho">10E-8</feature>
</node>
</model></td><td>Each node definition, regardless of the nesting structure in a different model, can be defined with the same id relating the node with one from a different model.

Example on the left includes the node “p001” - which is described using separate features in two different models.</td></tr>
<tr><td colspan="2">Comparing the features and structures of nodes in identically classified models</td></tr>
<tr><td><struct>
<model class="mechanical">
<feature id="l" unit="mm" />
<feature id="d" unit="mm" />
<node class="spigot" id="p001">
<feature id="l">48</feature>
<feature id="d">35</feature></td><td>Each feature and node, or structure of nodes can be compared with another one, only if they are defined in an identically classified model. The model itself also provides methods and appropriate practices for the comparison.

The suggested approach to take when comparing</td></tr>
</table>

<table>
<tr>
<td>

```
  </node>
</model>
</struct>


<struct>
<model class="mechanical">
  <feature id="l" unit="mm" />
  <feature id="d" unit="mm" />
  <node class="spigot" id="p002">
    <feature id="l">68</feature>
    <feature id="d">32</feature>
  </node>
</model>
</struct>
```

</td>
<td>the codes is to match them based on the nesting level and its class and then features.

Notice: the features can vary in terms of used units, but automatic normalization of units can be attempted for a given classification model. This is enforces the need for pre-calculation and verification of the model before further numerical analysis.</td>
</tr>
<tr>
<td colspan="2">Referencing the feature values from different models</td>
</tr>
<tr>
<td>

```
<struct class="shaft" id="s001">
<model class="geometrical">
  <feature id="S" unit="m^2" />
  <feature id="V" unit="m^3" />
  <node class="cylinder" id="p001">
    <feature id="S">
      PI*(p001.d/2)^2</feature>
    <feature id="V">
     p001.S*p001.l</feature>
  </node>
</model>
<model class="mechanical">
  <feature id="l" unit="mm" />
  <feature id="d" unit="mm" />
  <node class="spigot" id="p001">
    <feature id="l">48</feature>
    <feature id="d">35</feature>
  </node>
</model>
<model class="electrical">
  <feature id="rho" unit="Ohm*m" />
  <feature id="R" unit="Ohm" />
  <node class="conductor" id="p001">
    <feature id="rho">
      10E-8</feature>
    <feature id="R">
     rho*p001.l/p001.S</feature>
  </node>
</model>
</struct>
```

</td>
<td>Since the values of the features can contain functions to be evaluated, and by so are implicitly linked to values of other features, in possibly different units, and classification models, this should be done with high diligence.
The analysis performed on such defined structures is strongly suggested to be first pre-calculated and verified before applying further numerical or lexical processing methods.

Example on the left presents a more complicated, but allowed by the language, structuring of the model crossing dependencies.
Such description, after parsing yields a structure with its basic mechanical, electrical, and geometrical features linked with each other in one entity “p001”.</td>
</tr>
</table>

The concept of nodes, through their CLASS attribute, allows forming abstract relations that the designer wants include in the structure. The features defined directly in such abstract class are considered available for it, still the main data carrying concept is

the overall structure of nodes, even if further parametrized via the composition (or other) node. This approach, in addition to the language defined, adds further flexibility into the method enabling inclusion of abstract know-how into the description.

Approach taken by this method, allows inclusion of further enhancements to the structure with the ability to define own classification models and classes of nodes providing a unique set of functions enabling interpretation of the representation of the structure. To illustrate this approach, let's take the abstract node representing the milling direction into consideration. Such node, grouping several spigots of the shaft, provides additional information to the description of the element being not only the measurement of its features. This approach also opens the possibility of defining compositions of elements or their more complex collections in form of abstract classes of nodes.

Table 4. Examples of using identifiers in the language

<table>
<tr><td colspan="2">Unique IDs and their usage in features and feature values</td></tr>
<tr><td><model class="mechanical">
<feature id="l" unit="m" />
<feature id="d" unit="m" />
<node class="spigot" id="p001">
<feature id="l">0.48</feature>
<feature id="d">0.15</feature>
<node class="mill" id="p001a">
<feature id="l">p001.l/8</feature>
</node>
</node>
</model></td><td>The ID attribute when used in the feature element, should be heavily reused (within the model it has been defined), and is required to know what feature is being measured in a containing it node, and what are the common and comparable measurements done on the element's nodes.

The values of the features normally containing textual values, can reference feature values of other nodes and form simple mathematical functions.</td></tr>
<tr><td colspan="2">Unique IDs and their usage in nodes and their co-identification role in the structure</td></tr>
<tr><td><model class="mechanical">
<feature id="d" unit="m" />
<node class="spigot" id="p001">
<feature id="d">0.15</feature>
</node>
</model>
<model class="geometrical">
<feature id="S" unit="m^2" />
<node class="cylinder" id="p001">
<feature id="S">
PI*(p001.d/2)^2</feature>
</node>
</model></td><td>The same ID attribute of the structure's node, represents a direct identity relation between the nodes, and can be repeated only in a different model linking the two (or more) node structures with each other.

The example on the left demonstrates, how the same node "p001" has different features defined in separate models. Additionally, there is an implicit relation between the features with the surface of the cylinder depending on the diameter of the spigot.</td></tr>
</table>

Usage of the value of the ID field in the CLASS of a KXML element (interpreted as inheritance) provides the language with a method of building a re-usable library of both features and nodes. Additionally, usage of this type of a relation for representation of the structures adds further depth to the description of implicit relations in the design.

As much as nodes or features with different values of the ID attribute represent different instances, no meta-data comes along the usage of common but abstract CLASS names. As mentioned earlier, reusing the same values of the CLASS supports comparability of features and nodes, but by using a defined ID as a CLASS name it is possible to form an inheritance tree including features or structures of nodes and features assigned to them.

4.2. FEATURE VALUES AS FUNCTIONS

Values of features can be either values or functions that when evaluated provide a value. If the value of the function cannot be evaluated (i.e. one of the references is incorrect) a NULL value will be returned for the given feature (same as when the value of the entity has not been provided). The functions available for usage to calculate the value will depend on their implementation in the parser, and as much as there are many mathematical libraries, they have to be selected with care, and considering that even different versions of the same library can yield different results. These vary due to using different calculation models, algorithms, assumptions, constants, and errors. This further enforces the need for a symbolic representation, with a requirement of adequate and standardized approach to analysis of the mechanical structures.

The ability to calculate the functions referencing values of other features, requires the parser to wait until the whole structure has been loaded. This approach makes it easier to write the symbolic representation, not having to take into consideration the ordering of the nodes and features.

4.3. PARSING AND VALIDATION

The main requirements for the implementation of parsing mechanisms for this symbolic structure is maintaining the structure of nodes and leaving the calculation of feature values as a last and optional step. Feature values, when represented as functions, allow analysis of the implicit relations between the nodes, while the concrete value lacks that information. Depending on the type of the analysis, either one or both of the states of the value can be useful and if possible, both should be maintained in memory for analytical purposes.

To adequately address the requirements of the proposed symbolical description, the parsing process should be executed as follows:

A. Locating and loading the file with either an XML or JSON parser. During this step it is assumed that a notation based validation will be performed by the library in use. Most modern programming frameworks provide access to such libraries, but in case one is not available it is also possible to use (and possibly integrate to automate the process) one of many tools available on-line.
B. Traversing loaded data (structure containing multiple models, defining features, and a structure of nodes with concrete feature values) to build an adequate (for the future

analysis) in-memory representation of the description, independent of the notation used to store it.

Traversing through loaded data should be done using defined keywords and following outlined structure of the symbolic language. This approach allows automatic filtering of any invalid data that might have been included in the file.

Additionally, an optional comparison of the number of identified nodes against the total number of nodes present in the file, provides a completeness metric for this process.

C. Integrity of the loaded structure should be validated against the rules for relations defined in the symbolic language. This phase should be further adjusted to the input requirements of the analysis, but the integrity of the structure can be tested with:

   a. completeness of provided data - verifying if all the required attributes are set, and the minimum data is provided for processing;
   b. correctness of feature relations - checking if used feature ids are defined in all cases;
   c. correctness of node relations - detection of non-unique (in model) IDs in the node structure, not-looped node's class trees;
   d. correctness of functions in feature values - successful parsing and loading into memory each of the functions along with the detection of stack loops. It is important to notice here, the possible differences in scope, extensibility, and algorithms used in math libraries;
   e. correctness of function based relations between features - verification of all references to other feature values, the existence of data that is referenced, and verification of units (optionally automatically normalizing them, by automatic wrapping in adequate functions).

D. As the last step of the process, all of the defined models for the structure should be integrated into one, in-memory, directed, graph, data structure. The integration of the models, consists of merging them taking node IDs as the reference identity points. For the merging to be successful, there cannot be any nodes disconnected from the structure (i.e. when there is no node IDs overlap between the models).

4.4. MINIMIZED NOTATION

While the in-memory representations of the symbolic description will mostly depend on used tools and programming languages, the stored representations (in addition to recommended XML or JSON notations) can of course be used. The possibility to define the same symbolical object-oriented description in other notations (i.e. RDF, YAML) can be also used to define highly customized notations for more efficient processing by humans or machines.

As an example of such custom notation, in this case attempting to increase the briefness and simplicity of the notation the shaft presented on Fig. 3 can be defined as:

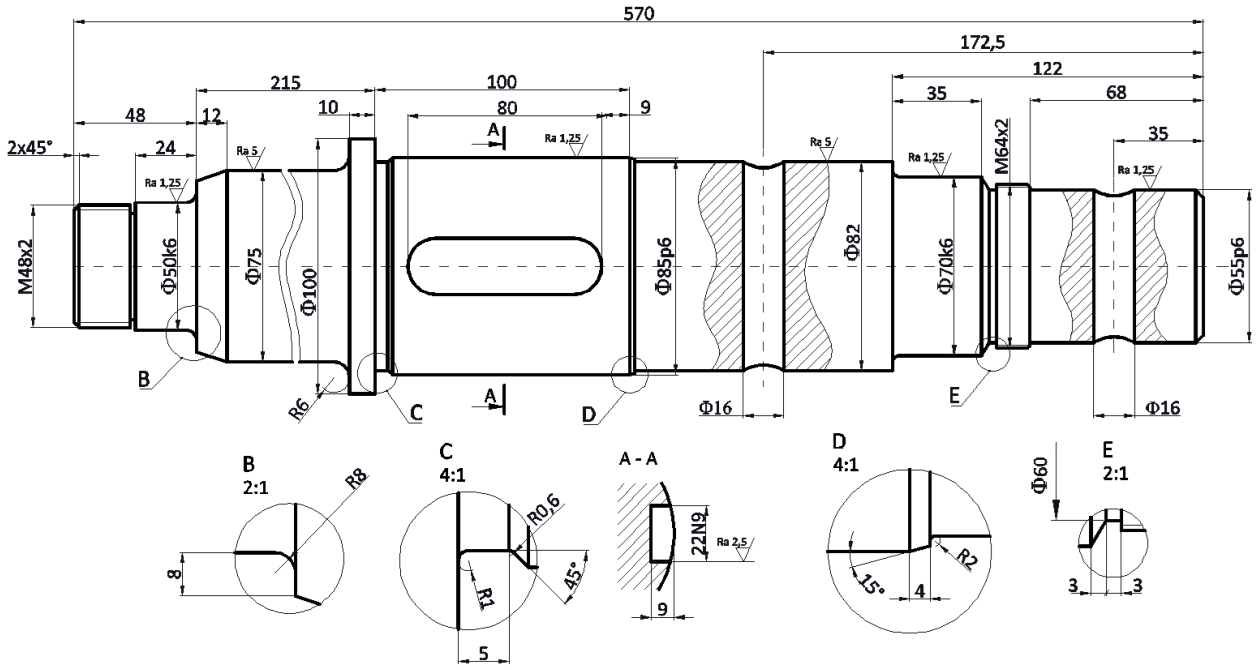


Fig. 3. Example of a shaft

Listing 1. Symbolic representation of a shaft

```
name = Fig. 2. Example of a shaft; class = shaft;
mechanical = {
  features = { l = mm; R = mm; d = mm; Phi = mm; id = PN; deviation = PN;
    Ra = { unit = µm; function = MRR } angle_y = { unit = deg; vector = (0,1,0) }
    angle_z = { unit = deg; vector = (0,0,1) } }
  nodes = {
    node1 = {
      class = spigot; features = { l = 48; Ra = 1,25; Phi = 50; deviation = k6 }
      nodes = {
        mill = { l = 2; angle_y = 45 } thread = { id = M48x2; l = 24 }
        undercut = { R = 8; d = 8 } }}
    node2 = {
      class = spigot; features = { l = 215; Ra = 5; Phi = 75 }
      nodes = { mill = { l = 12 } undercut = { R = 6 } spigot = { l = 10; Phi = 100
}}}
    node3 = {
      class = spigot; features = { l = 100; Ra = 1,25; Phi = 85; deviation = p6 }
      nodes = {
        groove = { Ra = 2,5; h = 9; angle_z = 90; l = 80; b = 22; deviation = N9 }
        undercut = { R = 1; b = 5 } mill = { angle_y = 45; undercut = { R = 0,6 }} }}
    node4 = {
      class = spigot; features = { Ra = 5; Phi = 82 }
      nodes = { mill = { l = 4; angle_y = -15 } undercut = { R = 2 } hole = { Phi =
16 } }}
    node5 = {
      class = spigot; features = { l = 122 }
      nodes = {
```

```
        node5_1 = {
          class = spigot; features = { l = 35; Phi = 70; Ra = 1,25; deviation = k6 }
          nodes = { undercut = {} }}
        node5_2 = {
          class = spigot; features = { l = mill.l + thread.l + mill.offset; Phi = 60
}
          nodes = {
            mill = { l = 3; angle_x = -60 }
            thread = { id = M64x2; l = node5.l-node5_1.l-node5_3.l-mill.l-mill.offset
} }}
        node5_3 = {
          class = spigot; features = { l = 68; Ra = 1.25; Phi = 55; deviation = p6 }
          nodes = { hole = { Phi = 16 } mill = {} }}}}}
geometrical = {
  features = { offset = { unit = mm }}
  nodes = {
    composition = {
      features = { vector = (1, 0, 0) }
      nodes =
        l2r = { class = composition; features = { vector = (1, 0, 0) }
          nodes = {
            node1 = {
              class = composition; features = { vector = (1, 0, 0)}
              nodes = { mill = {} thread = {} undercut = {} } }
            node2 = {
              class = composition; features = { offset = node1.l; vector = (1, 0, 0)
}
              nodes = { mill = {} undercut = {} spigot = {} } }}}
        r2l = { class = composition; features = { vector = (-1, 0, 0) }
          nodes = {
            node5 = {
              class = composition; features = { vector = (-1, 0, 0) }
              nodes = {
                node5_3 = { class = composition; features = { vector = (-1, 0, 0) }
                  nodes = { mill = {} hole = { offset = 35 } } }
                node5_2 = {class = composition; features = { vector = (-1, 0, 0) }
                  nodes = { thread = {} mill = { offset = 3 } } }
                node5_1 = {} }}
            node4 = {
              class = composition; features = { offset = node5.l; vector = (-1, 0, 0)
}
              nodes = { hole = { offset = 172,5-node5.l } undercut = {} mill = {}} }
            node3 = {
              class = composition;
              features = { offset = node5.l + node4.l; vector = (-1, 0, 0) }
              nodes = { groove = { offset = 9 } mill = {} undercut = {}} }}}}}}
```

5. CALCULATION MODELS

Having defined a concrete description of tested elements we can serialize the data contained in it to a form adequate for the qualitative algorithm we want to apply. Authors have not yet fully explored the capabilities of inclusion of the node graph in

the representation, but even simple models (1) provide quick and relatively detailed representation of the structure's quality.

$$\sum_{n=1}^{k} \frac{w_n}{\Delta d_n}; w_n \geq 1; \Delta d > 1; \tag{1}$$

for k compared features, where: w-weight, Δd-distance from reference feature.

The normalization of feature values in this case, due to the nature of the summation function used, is two-fold. First, the normalization of feature values has to provide an abstract for the differences in scale. This can be addressed by selecting derivative features (i.e. weight to length ratio or difference between angles) that have direct quality impact within the framework of the analysis. Second, each of the features included in the calculation has to normalized with a method with the same base (i.e. 1..10), so while there can be different functions used to normalize different features, the base value of each has to be identical. This assures that the impact of the calculated distance between features is equally weighted.

Furthermore, introduction of such normalization functions provides us with several opportunities. One is mapping of the correctness bounds to different values of the features. For example, measuring the collineation angle between the bolt and the slot, within the normalization function we can assume that all values above five degrees are incorrect, while collineation value of zero is correct. Another opportunity lies in the shape of the normalization function (Fig. 4.).

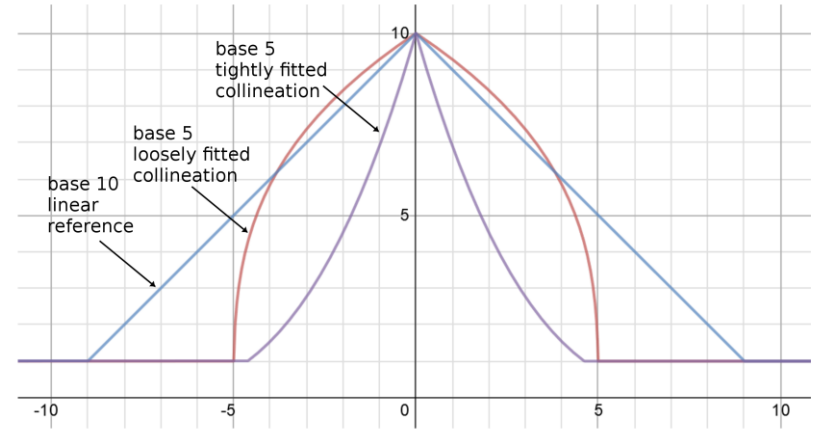


Fig. 4. Example feature value normalization functions.

This indicates, that bounded trigonometrical functions can be very useful in tuning the calculation of the feature distances. In particular, these modifiers can be used when there is a need to demonstrate the non-proportional nature of distances between feature values. For example, a 25% difference between a collineation value in an example range (0..5) can represent a "drop in quality" which will differ in both cases: 0 → 1 and 4 → 5. The red and purple values on Fig. 4. demonstrate this difference between tightly (purple) and loosely (red) fitted collineation features, both normalized to the same base 10 value, and both bounded to the same 0..5 range.

Another interesting approach lies in the possibility of representing the feature values as wave functions, yet the level of universality and meaningfulness of the interpretation of data depends on applied algorithm. Regardless of the approach, the normalization requirements have to be correlated with the method applied by the chosen analytical method.

Proposed antipattern matching factor (1), given normalized feature values, requires only value based normalization, and its calculation is done in three steps (Fig. 3.). At first we calculate the distances of the tested structure's feature values with their corresponding values in the reference antipattern. Notice, that these distances (Δd) have to be normalized with a base value shared among all features. The weights introduced in the second step provide a numerical way to change the impact of the feature's distance (Δd) on the final value. The third step is the summation of all of the weighted distances and comparing the results of different structures.

This relatively simple approach provides a quick method of comparing the quality of structures, yet it requires manual tweaking of the weights to emphasize specific features and their impact on the correctness of the relation. Additionally the necessary steps of selection of features that should be compared seems less practical.

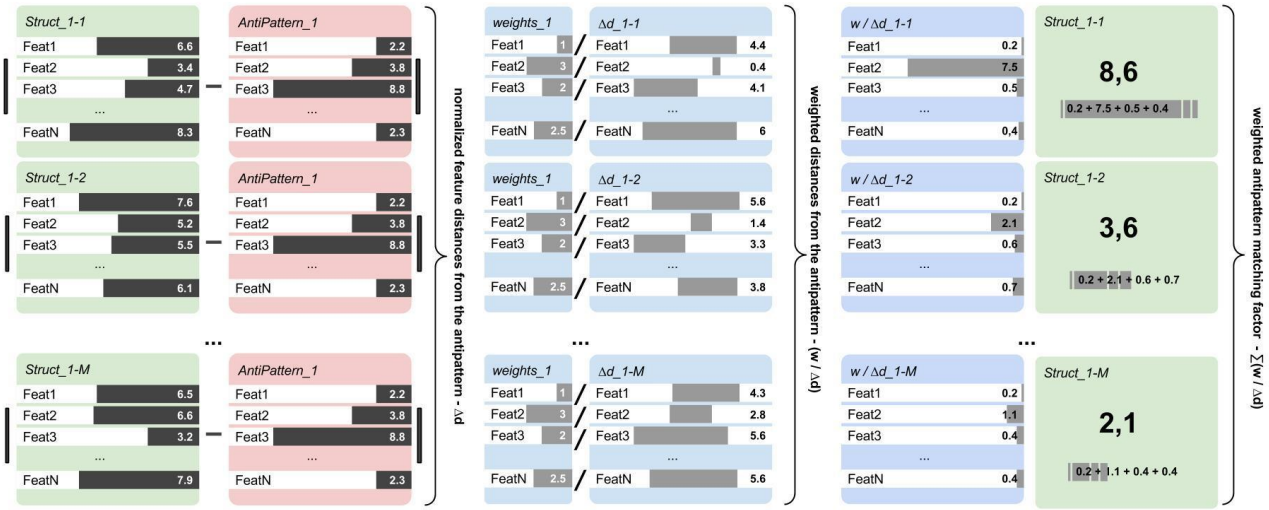


Fig. 5. The compatibility factor with antipattern

Further normalization that is suggested to efficiently process structural information, is limited to scaling and bounding of the input data and interpretation of the output data. The normalization process provides also an ability of inserting external requirements to analytical calculations. This is possible through tuning of the definition of the normalization ranges and normalization functions (Fig. 4.) and applying different normalization functions with different capabilities of representing the data.

For the purposes of a specific analysis, the values of a normalized, symbolic description have to be serialized in a consistent manner, so that the algorithms using it can make assumptions about the data’s structure. In addition to the simple arrays and binary

forms, we can present input data as a binary table (Fig. 6.) where rows list all processed features and the assignment of the active bit to one of the columns represent its value. Activated bit will depend on the magnitude of the normalized feature value. This approach can lead to precision loss that is dependent on the number of columns allowing differentiation between feature values, however a gradable level of precision provides potential advantages in speed of calculation without significant quality loss.

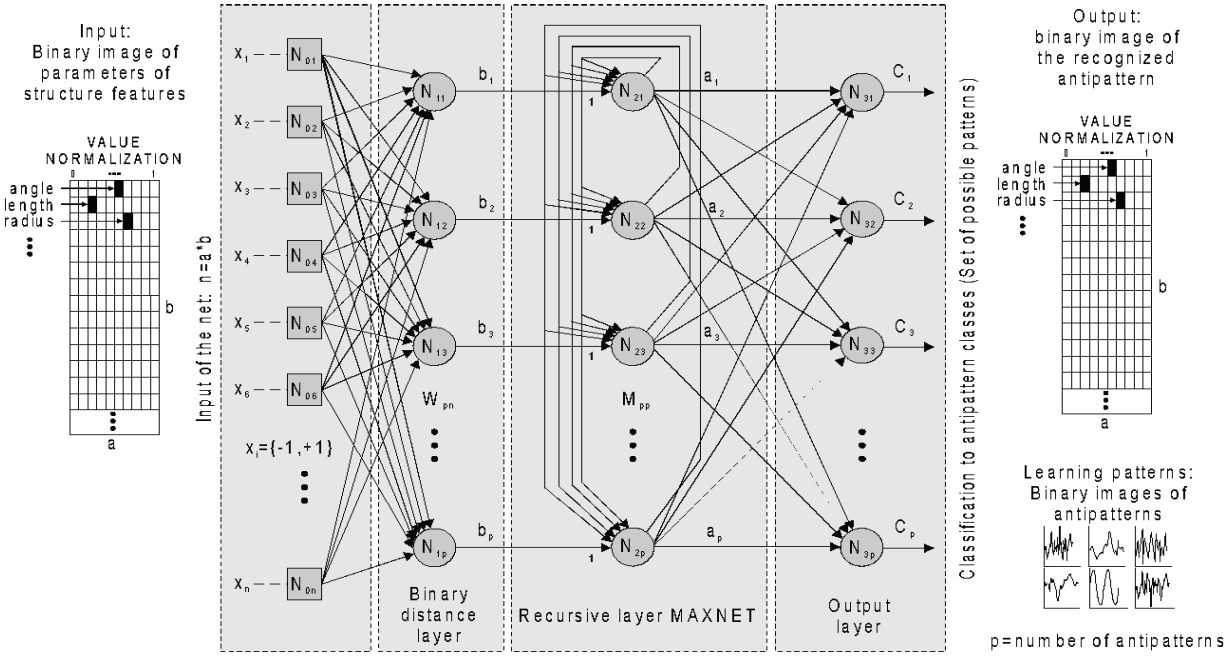


Fig. 6. A schema of applying a Hamming net to normalized feature values

Proposed normalization method enables usage of neural networks over the data describing mechanical design, by providing a common value format which can be compared, and on which patterns can be detected. Matching of such patterns (classification) with parts of other mechanical designs used to train the network, enables qualitative evaluation of the mechanical elements. These pattern matching capabilities will vary depending on the algorithm, but will in some way mimic the detection of similarities between data bits.

Having in mind the significance of the calculation models enabled by neural networks and machine learning algorithms, we propose usage of such normalized, symbolic representation of structural features with a Hamming net [10] (Fig. 6.) and a probabilistic network [13] (Fig. 7.). The benefits of applying neural networks arise largely from being able to handle much larger amounts and variations of input data, resulting in a more flexible but possibly less precise qualitative calculation model.

Applying a Hamming net to the feature data of the tested mechanical elements, the normalized, input, feature values (in range: 0..1) provide difference in signal only for *a* different states in each of *b* different features. The net is built upon the Hamming distance using the difference in the number of bits.

Each of the feature values is considered a bit, providing a value directly related to the differences observed in the structures. This means that as much as there is no direct valuation or grading between tested elements, the method provides a quick way of locating the most similar antipatterns. This can be observed as locating similarities in the tables (Fig. 6.) and making a quality decision based on the similarity to one of the antipatterns (possibly expanded by using Hamming's weight function).

In case of a probabilistic neural network (Fig. 7.), the model is calculating the probability to achieve the same goal of general classification of structures.

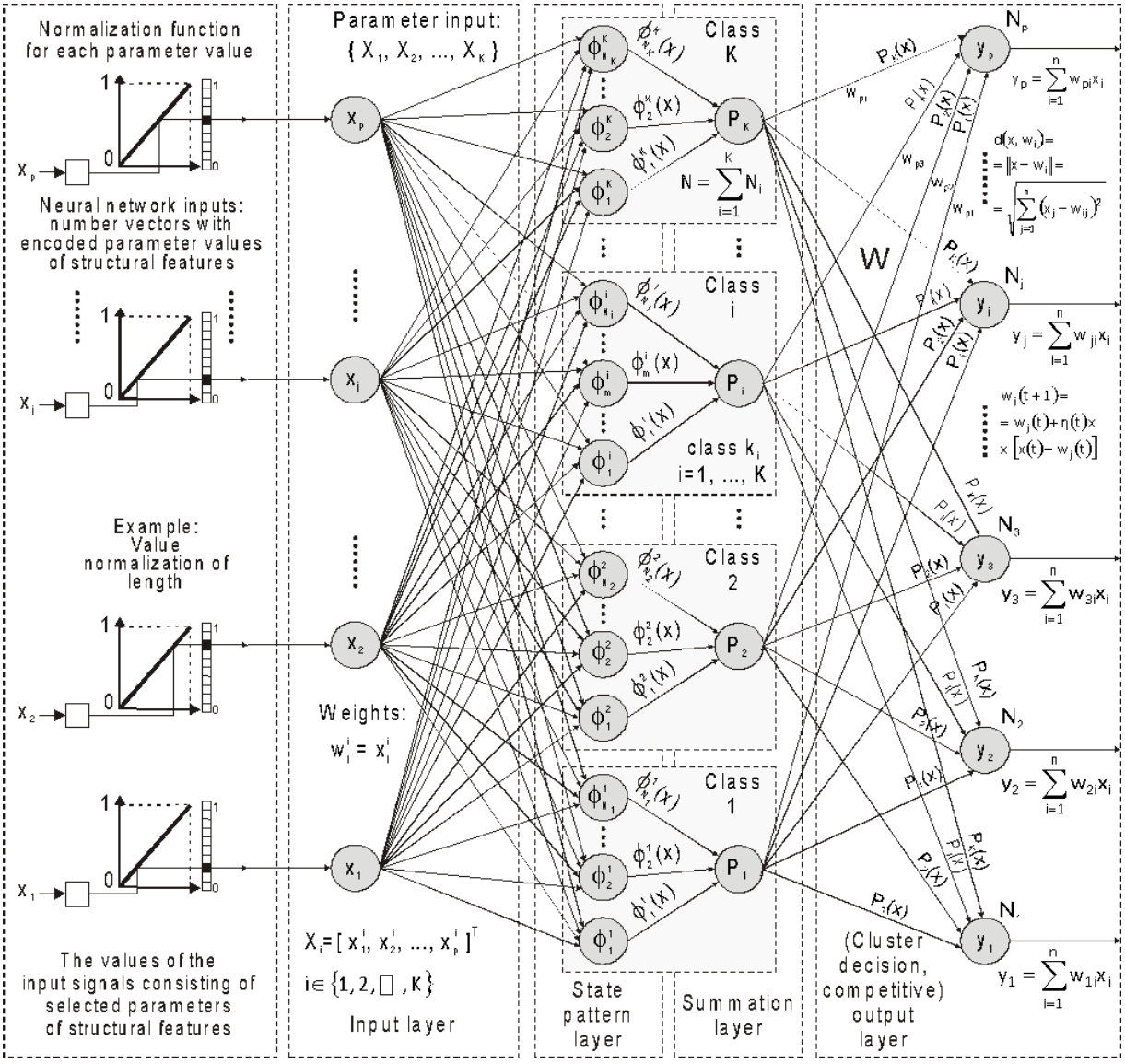


Fig. 7. A schema of applying a probabilistic neural network to normalized structures

The exemplified (Fig. 7.) probabilistic network requires the input data to be normalized in a similar manner as the Hamming net and will produce data describing similarities that can be interpreted in a similar manner as ones of the Hamming net's.

Both approaches (Fig. 6., Fig. 7.) enable classification of structures basing on their features. In the Hamming net, the distance between two compared structures (being the bias of the neuron function) is calculated as a Hamming distance (number of differing bits, in our case rows) between two serialized values. The precision of both these approach can also be further enhanced by taking into consideration the positive and negative interpretation of the distance, as the reference structure can be compared against an antipattern or a correct structure.

6. SUMMARY AND CONCLUSIONS

Proposed symbolic notation, the method of structural data normalization, processing and processing, provide a low cost, flexible, and universal way of benchmarking the structural quality of mechanical elements. This approach enables creation of IT systems that mechanical structure designers can use to incorporate common design knowledge enhanced by additional design quality requirements directly within their tool-set.

High flexibility and low cost of high scale calculations, enable usage of this method for verifications of automatically produced designs by genetic and machine learning algorithms. Furthermore, the symbolic object notation allows quick verification of completeness of the input data, such that it can be quickly compared (in an automated manner) with the required scope of input data for each class of used elements. As a side effect of replacing the graphical representation with a symbolic one, authors observed the avoidance of errors induced by overlapping or unreadable elements on the graphic.

Other potential uses of this method include the ability to easily translate structural descriptions between different metric systems, visual representation models and techniques. Additionally, the universal character of the symbolic representation of structures and their and antipatterns, hints that a similar approach can be applied in other disciplines of science.

An interesting direction of further research is the possibility of enabling automated structural analysis of machine elements through application of machine learning algorithms.

ACKNOWLEDGMENTS

This project was financed from the funds of the National Science Centre (Poland) allocated on the basis of the decision number DEC-2012/05/B/ST8/02802.

REFERENCES

[1] BROWN W.J., MALVEAU R.C., H.W. MCCORMICK, MOWBRAY T.J., 1998, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, Wiley, New York.

[2] KACALAK W., MAJEWSKI M., 2012, *New Intelligent Interactive Automated Systems for Design of Machine Elements and Assemblies*, ICONIP2012, Doha, Qatar. Lecture Notes in Computer Science, 7666, Part IV, Springer, 115-122.

[3] KACALAK W., MAJEWSKI M., 2014, *Interactive design of machine elements and assemblies*, Archives of Mechanical Technology and Automation, 34/3, 13-22.
[4] KACALAK W., MAJEWSKI M., TUCHOLKA A., 2015, *Intelligent Assessment of Structure Correctness Using Antipatterns*, International Conference on Computational Science and Computational Intelligence CSCI2015, Las Vegas, USA. IEEE Xplore Digital Library, 559-564.
[5] KACALAK W., MAJEWSKI M., 2015, *Interactive design of machine elements in uncertainty and unrepeatability*, Advances in Manufacturing Science and Technology, 39/2. 5-15.
[6] KACALAK W., MAJEWSKI M., STUART K., BUDNIAK Z., 2015, *Interactive Systems for Designing Machine Elements and Assemblies*, Management and Production Engineering Review, 6/3, 21-34.
[7] KACALAK W., MAJEWSKI M., BUDNIAK Z., 2015, *Intelligent Automated Design of Machine Components Using Antipatterns*, International Conference on Intelligent Data Engineering and Automated Learning IDEAL2015, Wroclaw, Poland. Lecture Notes in Computer Science, 9375, Springer, 248-255.
[8] KOENIG A., 1995, *Patterns and Antipatterns*, Journal of Object-Oriented Programming, 8/1, 46-48.
[9] LONG J., 2001, *Software reuse antipatterns*, ACM SIGSOFT Software Engineering Notes, 26/4, 68-76.
[10] MAJEWSKI M., ZURADA J.M., 2008, *Sentence Recognition Using Artificial Neural Networks*, Elsevier Knowledge-Based Systems, 21/7, 629-635.
[11] PIEGL L.A., 2005, *Ten challenges in computer-aided design*, Computer-Aided Design, 37/4, 461-470.
[12] RIEL, A.J., 1996, *Object-Oriented Design Heuristics*, Addison-Wesley, Reading, MA.
[13] SPECHT D.F., 1990, *Probabilistic neural networks*, Neural Networks, 3/1, 109-118.
[14] ZENG, Y., HORVATH, I., 2012, *Fundamentals of next generation CAD/E systems*, Computer-Aided Design 44/10, 875-878.