A DATA EXCHANGE NEUTRAL FORMAT FOR FINITE ELEMENT CODES USING AN INTELLIGENT ELEMENT CLASSIFICATION

Gray F. Moita[1] , Paulo Eduardo M. Almeida[2], Denise M.L.M. Oliveira[3], Marden C. Pinheiro[4]
[1]Centro Federal de Educação Tecnológica de Minas Gerais, E-mail: gray@dppg.cefetmg.br
[2]Centro Federal de Educação Tecnológica de Minas Gerais, E-mail: paulo@dppg.cefetmg.br
[3]Centro Federal de Educação Tecnológica de Minas Gerais, E-mail: deniseoliveira@civil.cefetmg.br
[4]Universidade FUMEC, E-mail: marden@fumec.br

ABSTRACT

This work presents the basis of the Finite Element Markup Language (FEML) proposal and also an intelligent tool for the recognition and classification of finite elements families used in conjunction with the above XML schema. The need for the classification becomes clear due to the necessity of a mapping between the several elements in a given code and the FEML format. A tree-type search could be used but it can lead to a lengthy if-then-else sequence to perform the matching. The use of a database-type classification could also be an option; however the search mechanism would still be required. Moreover, the use of an intelligent-type classification is also able to accommodate changes in the element families to be classified and new elements can be added with minimum effort. Here, the so-called intelligent classification uses the technology of artificial neural networks, specifically through of application the Self-organizing Map Networks and the Learning Vector Quantization. To carry out this task, the network receives the attributes of the elements and makes its classification, in terms of equivalent (or similar, less specialised) elements. At the end, it allows for the interface programs to "translate" between a specific (proprietor) format and the neutral format, in an intelligent way, minimizing errors due to manual data transcription and eliminating the necessity of searching the entire tree of elements. **Keywords:** Classification; finite elements; artificial neural network; self-organizing map; learning vector quantization; XML schema.

1. INTRODUCTION

Structure analysis software that use the Finite Element Method (FEM) are programs that handle physical phenomena which are expressed by means of fairly complex models and tend to output great volume of data within their descriptions. Also, the difficulty of data interchange between FEM programs is notorious, since each of the programs features particular sets of elements, originated from distinct formulations, frequently used for the solution of one particular problem. The lack of a standard format for data interchange between these programs is notable and makes the multi-application work to solve or to model such a problem much harder.
The purpose of this present work is to ease the information exchange between several of the existing finite element programs. For this, one could make use of a standard output to feed other systems' input, thus maintaining data integrity and turning their processing into a more dynamic event.
In the field of Information Systems, the process of importing and exporting data has been facilitated by the usage of the Extensible Markup Language (XML). This language is a natural candidate to allow for the creation of a vast lexicon for the description of both data and structures to be used on the input and output interfaces of each FEM system. Its usage has been growing considerably in environments in and outside the Internet, as a basis for languages and vocabularies for the conception of neutral, non-proprietary and extensible formats, aiming at the electronic data interchange between distinct applications. Within this context, the Finite Element

Markup Language (FEML) scheme for data interchange between finite element software is an innovative and flexible proposal to the solution of such problems, since these programs work with families of elements, which can be mapped, at their majority, into equivalent elements in all other programs [1].
The usage of the Artificial Neural Networks (ANN) technology in the resolution of problems by classification has been widely spread. Some of the typical examples are voice, characters and image recognition [2]. Other examples are in complex systems capable of capturing important statistically intrinsic characteristics within an input space (data), describing collective sorting phenomena from a given initial disorder and forming groupings of similar patterns in a map of self-organizing characteristics.
The current work presents a brief description of the FEML tool, proposed by Pinheiro [3], which is a proposition of a neutral format for the structuring and data interchange between FE programs. The FEML was created with the aid of the XML and XML schema technologies. It also presents a tool, constructed with ANN, specifically by self-organizing networks and a Learning Vector Quantization (LVQ) algorithm, to be run in conjunction with the FEML. The network allows for the interface software between the many existing FEM programs to execute the data interchange operations in an intelligent and dynamic way from the FEML itself.

2 . THE FINITE MARKUP LANGUAGE – FEML

The XML was initially suggested as a complement to the HTML language, given the increasing demand for new tags, formats and data types available on the web. Connolly et al. [4] had already pointed out such demand, even before the first formal W3C (World Wide Web Consortium) recommendation, in 1998. Nowadays, the XML has rapidly grown to the point of holding a much higher level than that of a mere data displaying language for the Internet [5].
As mentioned above, the present work deals with the possibility of creating a XML vocabulary for the finite element data interchange applications. In general, these kinds of vocabularies are considered XML applications, created from a document model verifiable by any given syntactic analyser. From this concept, a way of validating an XML document can be created, once its structure and its element possible content are known. In this way, there is still the possibility of validating the content and the structure of a given document from a descriptor, creating, thus, a family of documents which complies with that determined descriptor. These families of documents many times gain their own denominations, like the MathML (Mathematical Markup Language), destined to the description of mathematical formulas and equations, the CML (Chemical Markup Language) for the description of chemistry formulas, the MatML (Material Markup Language), which characterizes materials, or the FEML(Finite Element Markup Language) itself, also an object of this study. The FEML is a XML Schema [6, 7].
The main goal of the FEML is to produce a vocabulary which described syntactic and semantically the data structure required by a FEM program, so that its export and import tools could be facilitated. In this way, a structure that has been modelled by a given program can have its data exported to another program using a neutral format. This avoids data re-entrance, a step that can take way too much time, given the amount of data in a typical model, and may also add typing errors. The proposal of this tool is illustrated in Figure 1, which also indicates another of the focus of the present study, that is, the application of ANN.

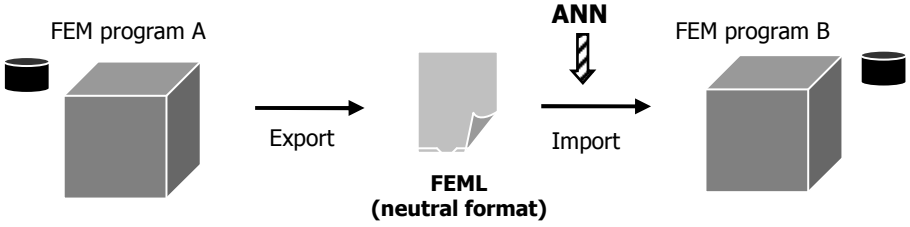

Figure 1: FEML's primary proposal

Most programs generate exportation files, but they do it in proprietary formats. Thus, to obtain the portability between these programs, each one should have several importation interfaces, basically one for each exporting program. As the number of programs increases, the viability of this approach weakens. Therefore, the creation of a standard export file becomes fundamental when using more than one program to run a specific analysis. This format could be shared between the software developers. However, it would be difficult to come up with a format which gathered all of the XML intrinsic facilities, such as its capability of syntactic and semantic description, as well as its hierarchical structuring, allied to the power of a Validation Scheme with the strong type setting and definition of content of a XML Scheme.

With that in mind, Pinheiro [3] proposed the basic XML Scheme FEML, as well as a suggestion of the tree structuring for the primary elements used by the LUSAS commercial software [8], here used as the basis for the present development. The FEML, in its initial proposal, creates a document which should basically describe the finite element topology of the model, the material of each element and the loading and boundary conditions to which the model is subjected, alongside with some control variables. This information, along with options for the project documentation (author, date, source of information, etc), should be sufficient, so that a second MEF program could be totally fed with the necessary data for the calculation straight out of them, having to have only a few additional data supplied, such as processing options or comments.

To achieve this initial proposition, it was necessary to dispose the elements in a hierarchically structured way, so that each program would not take the element characteristic name, as it is known in its original environment, but its primary characteristics and, from these, be able to define which of its implemented elements could be used on that case. In other words, as possibly there is not a one-to-one equivalence between the many implemented elements in each environment, the element type setting must be done through its characteristics, not by specific identifications from its original environment. This shows that the adoption of a data portability standard demands the adoption of other standards, such as this basic tree for the element's hierarchicalisation. An example is shown below:

```
<Example Family="2D"
  Shape="Triangular"
  Application="static"
  Order="quadratic"
  ElementType="plane stress"
  Name="TPM6"/>
```

Thus, a tree was sought (Figure 2) which, from a basic grouping of families, was subdivided by their primary forms, their application, the order and, finally, the element type.

In this way, each program could implement this hierarchy and organise its elements through it. The data received by FEML could then be identified from within this hierarchy, thus sufficing to the program to import the data referring to the topologies, materials, boundary conditions and loading, relieving the user from the responsibility of inputting all the data for that environment.

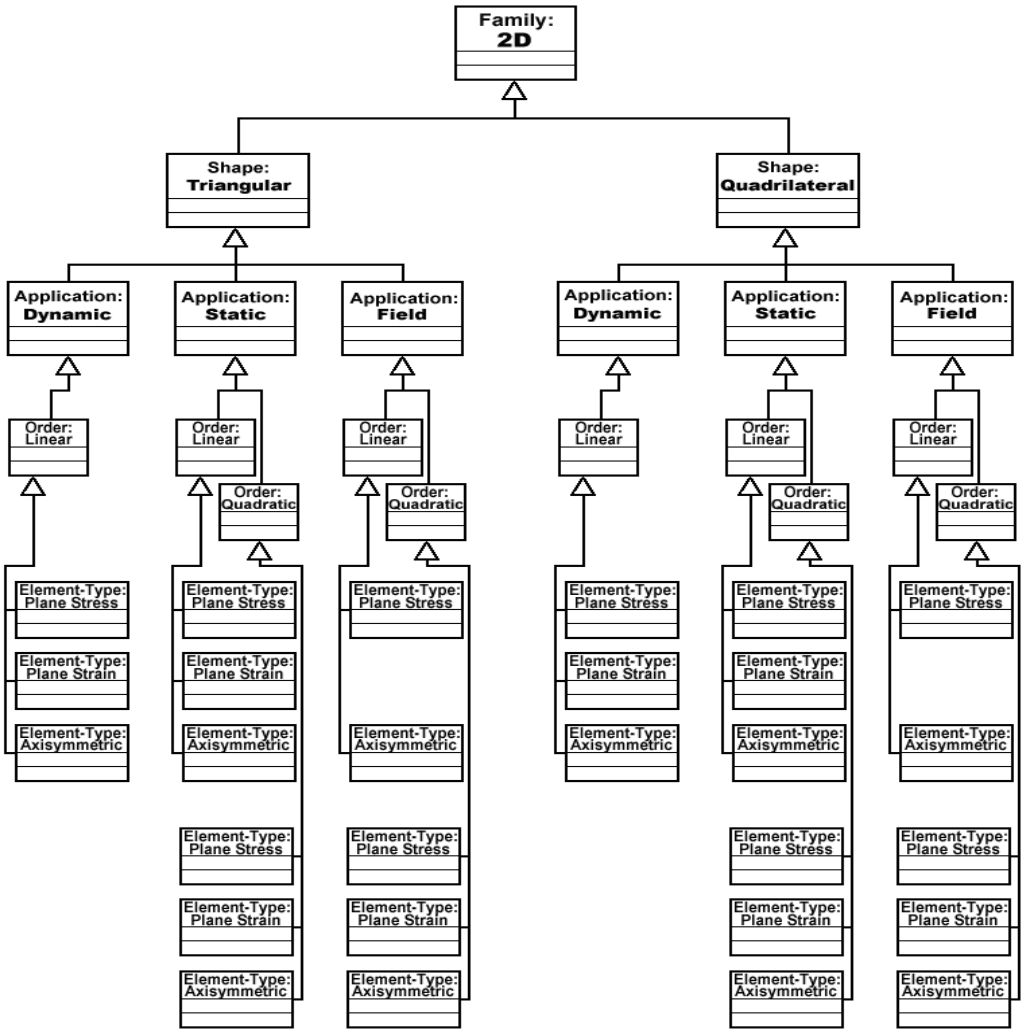

Figure 2: Continuous bi-dimensional space family hierarchy

3. ARTIFICIAL NEURAL NETWORKS

The ANN are mathematical models inspired after the biological neural structure and which possess computational capacity acquired by means of learning and generalization [2, 9]. They are constituted by a large number of simple processing elements, with many connections among themselves (synapses), which are prone to the storage of the experimental knowledge and to make it available for later use. The knowledge is acquired through the process of training.

The networks training methods can be arranged in two basic types: supervised learning (response monitoring) and unsupervised learning (unknown answer). Other two types are also very commonly used: reinforcement learning (the intermediate case between the supervised and the unsupervised learning methods) and the competition learning method (particular case of unsupervised learning).

On the supervised learning, the networks response is monitored by the supervisor. The learning set is formed by input and output pairs (x_i,y_{id}), where x_i represents the input pattern and y_{id} represents the expected output to the supplied x_i pattern. In this case, the network response to each x_i input is already known. The weight adjustment represented by Δw is done in such a way that the network y_i response to the x_i pattern approximates the desired y_{id} output.

As for the unsupervised learning method, the training is formed by the x_i input patterns and the weight adjustment, Δw, is only obtained through the values of the input pattern. The knowledge is acquired through the existing harmony with the statistical regularities in the network input pattern and through the redundancy of the data that it receives. This is how the network develops the ability to form internal representations to codify the characteristics perceived in the input and automatically creates new classes or groups.
The competition learning method is characterised by receiving the pattern from the "disputing" inputs, and only one of these will actually come out as the winner. These inputs are directly connected to the outputs, which can also be connected among themselves through lateral inhibitor (or negative) connections. The output with the greatest initial activation will have the best chance to score the dispute with the other outputs and these will, along the time, lose their inhibition power over the output with the biggest activation, rendering them completely inactive – except the winner. This method is known as Winner-Takes-All.
The solution of problems through classification in ANN occurs through the attribution of patterns such as the input to be identified from within a set of previously known classes. On the other hand, the solution by categorisation happens when the network finds some harmony with the statistical regularities of the input set, being, thus, able to group patterns through the codification of the supplied characteristics.
In this work, the results of three different network models, built to perform pattern classification using the supervised and unsupervised learning paradigms, are presented. The first case uses the supervised learning method, in which the input and output vectors are previously presented to the network, which performs their respective association. On the second case, the information is supplied through the set of patterns which the network receives as input, from which it then self-organises, defining its own parameters without any external aid, and that forms the concept of unsupervised learning. On the third case, the supervised algorithm corrects the parameters defined by the unsupervised method, aiming a gain in the overall performance of the network for the purpose of classification.

3.1 – Multi-Layer Perceptron – MLP

The MLP networks are characterised by presenting a unidirectional signal flux, one input layer, one or more layers hidden and an output layer. One of its primary advantages is the capacity of approximation of non-linear functions.
Learning in an MLP network occurs through a set of training patterns, composed of input arrays and their respective expected outputs, which will be presented to the network. The output and the error between the expected and obtained outputs are calculated, enabling the tuning of the patterns.
The training algorithm for the MLP networks is built as described in Table 1.

3.2 – Self-organizing Map – SOM

The SOM network was developed by Teuvo Kohonen during the eighties. It is neuro-physiologically inspired and was primarily based on the topological map present within the cerebral cortex. The neurons have lateral connection forces, which depend on their distances. This topological ordering is the result of the usage of lateral "refeeding" between the cerebral cortex cells [2].

Table 1: The training algorithm for MLP networks

1. Initialisation
    1.1 randomly initialise the w_{ij} weights*;*
    1.2 initialise the threshold variables θ_i;
2. Activation
    2.1 process the MLP network by the application of the x_i inputs;
        2.1.1 calculate the y_i outputs *;*
        2.1.2 increment the *1 (p=1)* iteration;
        2.1.3 calculate the outputs of the hidden layer for the *j* neuron;
        2.1.4 calculate the MLP network's output for all of its neurons.
3. Weight adjustments in the output layer
    3.1 calculate the error gradient committed in the current iteration for the neurons in the output layer;
    3.2 update the weights of the output layer.
4. Weight adjustments in the occult layer
    4.1 calculate the error gradient committed in the current iteration for the neurons in the occult layer;
    4.2 update the weights of the hidden layer.
5. Iterative repetition
    5.1 increment the *p* iteration counter;
    5.2 return to step 2.

The lateral refeeding of a SOM network is modelled by a popularly known function named the Mexican Hat. According to this function, each neuron influences the activation states of its neighbours in three ways: excitatory (close neighbours – within the ray ranging from 50 to 100 μm), inhibitory (within a second intermediate area) and lightly excitatory (most external area – ranging from the ray of 200 to 500 μm) [2].

The set of patterns supplied as the network input – which correspond the network nodes – is organized in a grid, generally bi-dimensional. Each node in the network receives all the inputs which are also connected to the neighbouring nodes (closest ones). The more similar the weight vector's input from a determined node, the bigger its output value will be. During the learning period, the nodes specialize themselves to the detection of a set of input patters; in other words, the nodes are topologically organised throughout the network, making the patterns detected by a given node relate to the coordinates of the node on the grid. Hence, the map of auto-sorting characteristics about the input patterns is formed, whereby the similar patterns are detected as being the closest nodes inside the grid.

The SOM training algorithm is built through competition. When a network node wins the competition, not only itself, but its neighbouring nodes will have their weights adjusted, a similar result as that of the Mexican Hat function.

The functioning of the SOM network happens basically when a certain pattern *p* is presented. The network will then search for *p*'s most similar node. During the training, the network magnifies the resemblance of the chosen node and of its neighbouring nodes to *p*. In this way, the network builds the topological map, where the neighbouring nodes respond similarly to similar input patterns.

A node activation state is determined by the euclidian distance between the weight and the input patter, as described in the Equation (1):

$$y = \sum_{i=1}^{n} \sqrt{\| x_i - w_{ji} \|^2} \qquad (1)$$

in which x_i is the input pattern, w_{ji} is the connection weight, n is the quantity of nodes in the network.

The neighbouring criteria shows how many nodes around the winner node will have their weights adjusted, that is, where the influence area of the winner node is defined. The neighbouring region may present different magnitudes and forms. Generally the forms used are linear, square, hexagon, circle and others, being the square the most commonly used. The definition of the most adequate form depends on the problem in question and on how the data will be distributed, normally defined by trial and error.
The size of the neighbourhood around the winning node is changed throughout the training process; it is initially wide, but is progressively reduced until the predefined limit, where the reduction rate is the linear function of the number of cycles. After the training, the network forms groupings, which will be labelled to indicate the class which they represent and thus allowing for classifications of unknown patterns.
The Equation 2 refers to the weight updating of the winner node and its neighbours.

$$w_{ji}(t+1) = \begin{cases} w_{ji}(t) + \eta(t)(x_i(t) - w_{ji}(t)), & \text{if } j \in \Delta(t) \\ w_{ji}(t), & \text{otherwise} \end{cases} \quad (2)$$

in which w_{ji} is the connection weight in the moment t, between the input element $x_i(t)$ and the j node and $\eta(t)$ is the learning rate.
Table 2 shows the training algorithm for the SOM networks.

Table 2: Training algorithm for the SOM networks

1. Initialisation
    1.1 initialise all the weights and parameters
2. Iterative repetition
    2.1 for each *x* training pattern do
        2.1.1 define the winner node;
        2.1.2 update the winner node and its neighbour weights;
        2.1.3 if the number of the cycle is a multiple of N
            then reduce its learning rate along with its neighbours rates.
3. Until the characteristics map does not suffer significant changes.

3.3 – Learning Vector Quantization – LVQ

The LVQ algorithm is a supervised learning technique which uses the information about the classes to lightly move the weight vector, aiming at the increasing the quality of the classifier's decision regions for each supplied pattern as an input to the network. That is, because it is supervised, the LVQ algorithm can, after the SOM training, evaluate the classification generated by the network for each pattern [10, 11]. It is through this evaluation that the LVQ algorithm proceeds, adjusting the weights, so that there can be a better classification of the pattern.
The adjustment of the map of characteristics can be seen as the first of two stages – using the SOM network – to adaptively solve a problem of classification by patterns. The second stage will use an LVQ algorithm to perform an even finer tuning of the map of characteristics.
The training occurs when some pattern x is randomly obtained from the input space. Thereafter, if the class labels of the input pattern x e and if the weight vector w are in conformity, then the weight vector is moved towards the pattern x. Otherwise, the weight vector is moved away from the pattern x.
During the classification, the LVQ algorithm sets the weights of the winner node and those of its neighbours, making the comparison between each of the network's input and output pattern and the expected output.

$$w_{ji}(t+1) = \begin{cases} w_{ji}(t) + \eta(t)(x_i(t) - w_{ji}(t)), & \text{correct class} \\ w_{ji}(t) - \eta(t)(x_i(t) - w_{ji}(t)), & \text{incorrect class} \end{cases} \quad (3)$$

where $\eta(t)$ is the learning rate.

Table 3 briefly describes the training steps according to the LVQ algorithm.

Table 3: LVQ training algorithm

1. Initialisation
    1.1 initialise all the weights and parameters
2. Iterative repetition
    2.1 for each *x* training pattern x do
        2.1.1 define the winner node;
        2.1.2 update the winner node and its neighbours weights;
        2.1.3 decrease the learning rate.
3. Until the error is smaller than a given value.

4. CLASSIFICATION MODELS

As mentioned, the purpose of the FEML is to facilitate the data interchange between the many existing FEM programs, by using hierarchical models to map a set of elements, typical of each family and present in the interface programs developed by many manufacturers, according to their basic characteristics. From the hierarchical models, interface sub-routines must be created by the finite element programs using the FEML to allow for the data importation and exportation processes.

4.1 – FEML Updating

With the goal of increasing FEML applicability and robustness, the inclusion of other families of elements to the model proposed by Pinheiro [3] and Moita and Pinheiro [1] was necessary. Consequently, the proposed model was updated with the addition of a continuous tri-dimensional space family and the complementation of the bi-dimensional continuous space family with the addition of new elements which have not yet been contemplated [10, 11]. As a result a new version of the hierarchical model is achieved, as depicted in figures 3 and 4.

The current hierarchical model had an increase of 225% over the previous one, which represents only three of the ten families within the LUSAS commercial program. Naturally, this hierarchy tends to grow even further with the addition of the other families. Therefore it is believed that a lot of processing time will be spent with the search routines to move from generalisation (the element family) towards specialisation (the element classification), so that the equivalent (or similar) element to the searched one can be found. Hence, the implementation of intelligent routines becomes necessary, to allow a more dynamic access to the specialisations from the generalisations.

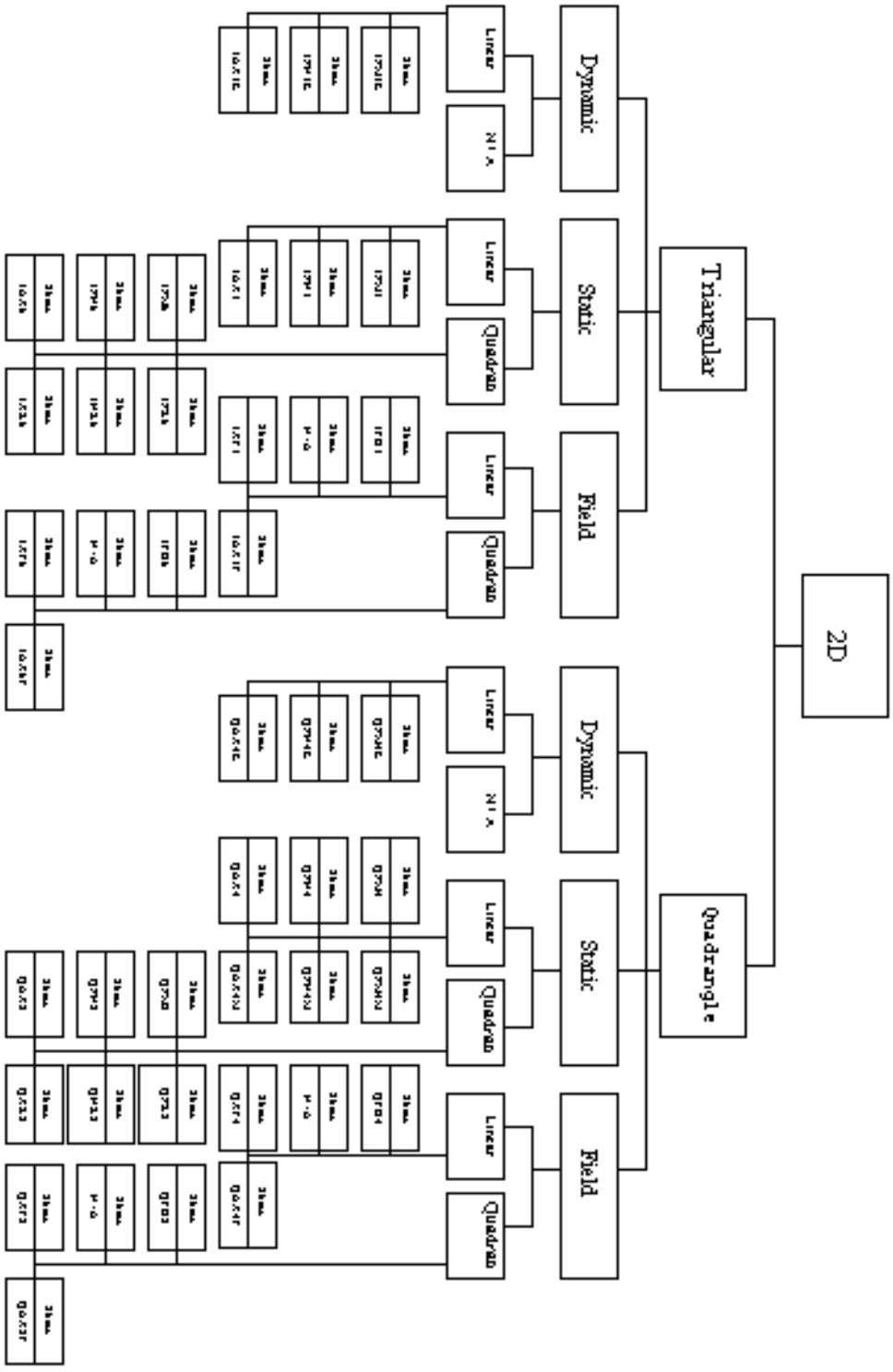

Figure 3: The bi-dimensional family hierarchy

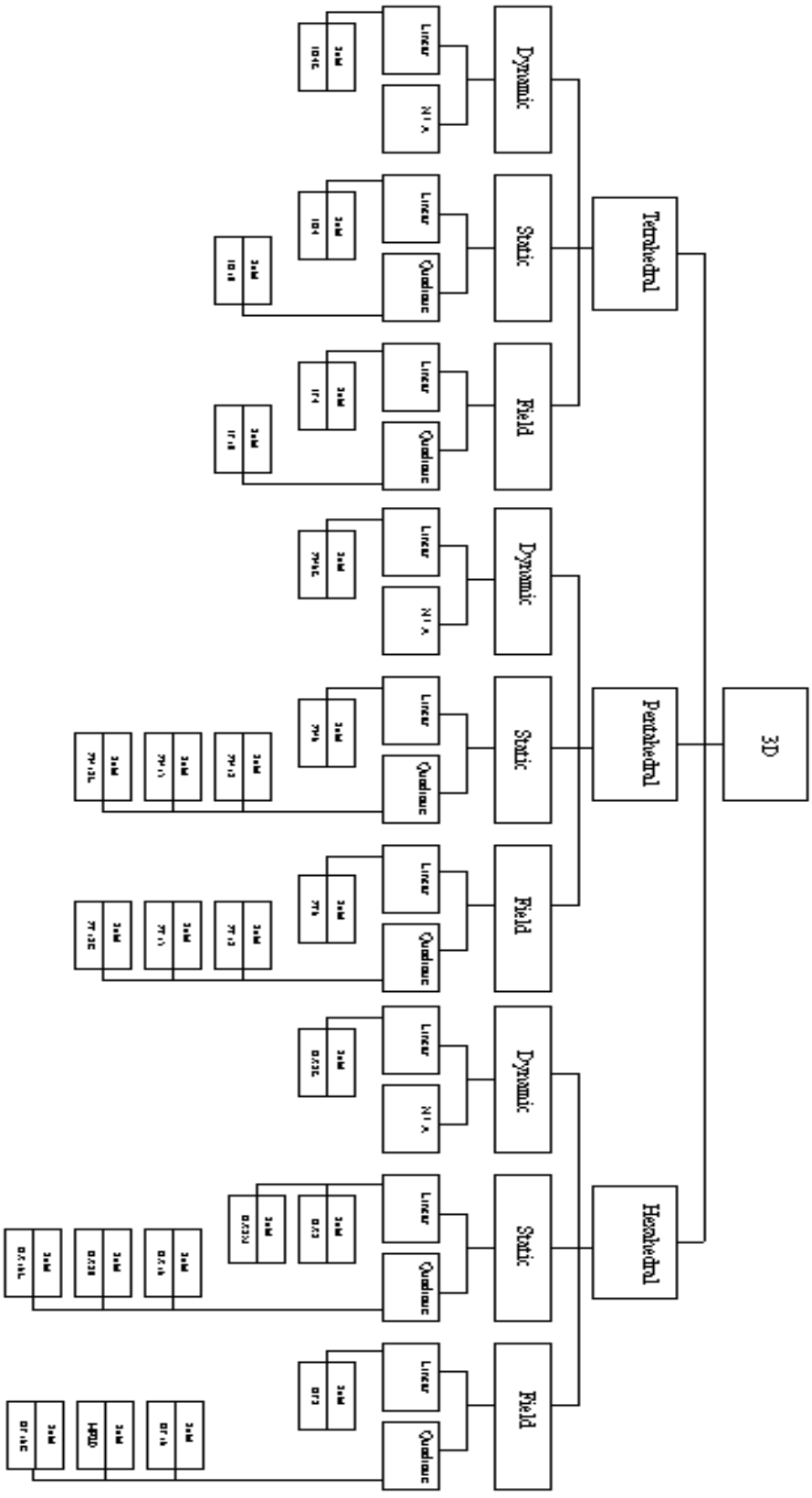

Figure 4: The tri-dimensional family hierarchy.

4.2 – Codification of the Finite Elements Characteristics

The data that feeds an ANN and which represents the set of its supplied patterns is numeric. Therefore, it was necessary to numerically codify the basic characteristics of the elements to be supplied as the input for the network constructed. The array *x* that feeds the network input has a dimension of five (5), which describe the element in a more orderly way: Family, Shape, Application, Order and Type, according to tables 4 to 6. Further ahead, Table 7 was created to perform the associations between the input patterns and the name of the element that is to be identified after its classification with the desired values of the output *y* [10].

Table 4: Codification of the families of elements

Cod_1	*Family*
2	Bidimensional (2D)
3	Tridimensional (3D)

Table 5: Codification of shape, application, order and element type

Cod_2	*Shape*	*Cod_3*	*Application*	*Cod_4*	*Order*	*Cod_5*	*Element_Type (2D)*	*Element_Type(3D)*
1	Triangular	1	Dynamic	1	N/A	1	Plane Stress	Solid
2	Quadrilateral	2	Static	2	Linear	2	Plane Strain	Solid
3	Tetrahedral	3	Field	3	Quadratic	3	Axisymmetric	Solid
4	Pentahedral							
5	Hexahedral							

Table 6: Codification of the element names

Cod	*Name*	*Cod*	*Name*	*Cod*	*Name*	*Cod*	*Name*	*Cod*	*Name*	*Cod*	*Name*	*Cod*	*Name*	*Cod*	*Name*
1	N/A	10	TAX6	19	N/A	28	QAX4	37	QXK8	46	N/A	55	PN12	64	HX8
2	TPM3E	11	TPK6	20	TXF6	29	QPM4M	38	QFD4	47	TH4E	56	PN15	65	HX8M
3	TPN3E	12	TNK6	21	TAX6F	30	QPN4M	39	N/A	48	TH4	57	PN12L	66	HX16
4	TAX3E	13	TXK6	22	N/A	31	QAX4M	40	QXF4	49	TH10	58	PF6	67	HX20
5	TPM3	14	TFD3	23	QPM4E	32	QPM8	41	QAX4F	50	TF4	59	PF12	68	HX16L
6	TPN3	15	N/A	24	QPN4E	33	QPN8	42	QFD8	51	TF10	60	PF15	69	HF8
7	TAX3	16	TXF3	25	QAX4E	34	QAX8	43	N/A	52	N/A	61	PF12C	70	HF16
8	TPM6	17	TAX3F	26	QPM4	35	QPK8	44	QXF8	53	PN6E	62	N/A	71	HF20
9	TPN6	18	TFD6	27	QPN4	36	QNK8	45	QAX8F	54	PN6	63	HX8E	72	HF16C

The following notation is used the tables 4 to 7:

- Network input: Vector *x* – *F*: Cod_1 (Family); *S*: Cod_2 (Shape); *A*: Cod_3 (Application); *O*: Cod_4 (Order); *T*: Cod_5 (Element_Type).
- Network output: Vector *y* – *Class*: Identifying label of the set of grouped patterns.
- Classification: *Cod* – Identifying code of the element name in the classification; *Name* – Element name in the classification.

Next, three network models are presented: classEF, somEF and lvqEF, constructed with the goal of validating this present proposal. For the creation of these three programs, MatLab Version 6.5 (Neural Network Toolbox) [11] was used.

Table 7: Classification of the elements basic characteristics

F	*S*	*A*	*O*	*T*	*Class*	*Cod*	*Name*		*F*	*S*	*A*	*O*	*T*	*Class*	*Cod*	*Name*
2	1	1	1	1	1	1	N/A		2	2	2	3	6	10	37	QXK8
2	1	1	2	1	2	2	TPM3E		2	2	3	2	1	11	38	QFD4
2	1	1	2	2	2	3	TPN3E		2	2	3	2	2	11	39	N/A
2	1	1	2	3	2	4	TAX3E		2	2	3	2	3	11	40	QXF4
2	1	2	2	1	3	5	TPM3		2	2	3	2	4	11	41	QAX4F
2	1	2	2	2	3	6	TPN3		2	2	3	3	1	12	42	QFD8
2	1	2	2	3	3	7	TAX3		2	2	3	3	2	12	43	N/A
2	1	2	3	1	4	8	TPM6		2	2	3	3	3	12	44	QXF8
2	1	2	3	2	4	9	TPN6		2	2	3	3	4	12	45	QAX8F
2	1	2	3	3	4	10	TAX6		3	3	1	1	1	13	46	N/A
2	1	2	3	4	4	11	TPK6		3	3	1	2	1	14	47	TH4E
2	1	2	3	5	4	12	TNK6		3	3	2	2	1	15	48	TH4
2	1	2	3	6	4	13	TXK6		3	3	2	3	1	16	49	TH10
2	1	3	2	1	5	14	TFD3		3	3	3	2	1	17	50	TF4
2	1	3	2	2	5	15	N/A		3	3	3	3	1	18	51	TF10
2	1	3	2	3	5	16	TXF3		3	4	1	1	1	19	52	N/A
2	1	3	2	4	5	17	TAX3F		3	4	1	2	1	20	53	PN6E
2	1	3	3	1	6	18	TFD6		3	4	2	2	1	21	54	PN6
2	1	3	3	2	6	19	N/A		3	4	2	3	1	22	55	PN12
2	1	3	3	3	6	20	TXF6		3	4	2	3	2	22	56	PN15
2	1	3	3	4	6	21	TAX6F		3	4	2	3	3	22	57	PN12L
2	2	1	1	1	7	22	N/A		3	4	3	2	1	23	58	PF6
2	2	1	2	1	8	23	QPM4E		3	4	3	3	1	24	59	PF12
2	2	1	2	2	8	24	QPN4E		3	4	3	3	2	24	60	PF15
2	2	1	2	3	8	25	QAX4E		3	4	3	3	3	24	61	PF12C
2	2	2	2	1	9	26	QPM4		3	5	1	1	1	25	62	N/A
2	2	2	2	2	9	27	QPN4		3	5	1	2	1	26	63	HX8E
2	2	2	2	3	9	28	QAX4		3	5	2	2	1	27	64	HX8
2	2	2	2	4	9	29	QPM4M		3	5	2	2	2	27	65	HX8M
2	2	2	2	5	9	30	QPN4M		3	5	2	3	1	28	66	HX16
2	2	2	2	6	9	31	QAX4M		3	5	2	3	2	28	67	HX20
2	2	2	3	1	10	32	QPM8		3	5	2	3	3	28	68	HX16L
2	2	2	3	2	10	33	QPN8		3	5	3	2	1	29	69	HF8
2	2	2	3	3	10	34	QAX8		3	5	3	3	1	30	70	HF16
2	2	2	3	4	10	35	QPK8		3	5	3	3	2	30	71	HF20
2	2	2	3	5	10	36	QNK8		3	5	3	3	3	30	72	HF16C

4.3 – Description of the ClassEF Network

The goal here is to perform the classification of the finite element through the attribution of the input patterns to be identified from within a set of previously known classes.

4.3.1 Implementation of the classEF

The classEF.m program is a network of the MLP type. It uses the backpropagation supervised learning method, based on the approximation between the ideal known behaviour and a real adjustable behaviour, through a set of input data and their respective classifications.
The following data was supplied to the network: a set of patterns representing the finite elements according to the codification described in Section 5.2 to the network input vector *x* and a set of classes to the vector *y* as an expected output to this network – also codified according to Table 7 – to be associated after the training of the initially informed vector *x*. Thus, this network was created containing five input elements, thirty intermediate layers and five output elements.

During the training stage, the learning rate was initially high, close to one (1), and it is gradually decreased until it reaches a given acceptable and previously informed value. In this case, up to five hundred (500) iteration cycles were necessary, so that this network could reach a learning rate of the expect order of 0.01, which is equivalent to 1 percent (1%) of errors in the result during the execution of the training stage.

4.3.2 Results of the classEF

The tests performed to verify this network model presented a one hundred percent (100%) satisfactory solution, once the input and output were previously informed and the trained network merely related the input and output vectors, in a one-to-one basis, which ought to be expected for a supervised network. In other words, the network was capable of associating the patterns it knew to the corresponding classes, which were already know as well. In the opposite side, the network was incapable of associating the new and unknown patterns with a representative class, in accordance to the resemblances between the known and trained patterns, presenting a result of zero percent (0%) hits on the unknown elements.
It was verified that this network presented, to the problem in question, a limitation facing an unknown answer to a new pattern supplied as its input. It was concluded that this network could not classify a new pattern by the similarities with the other already known patterns. Due to this limitation, the somEF model, described in the following section, was tested.

4.4 – Description of the SomEF Network

The goal now is to perform the classification of the finite elements, from which all that is known are the input data, forcing the network to find the class that is most similar to the set of supplied data. Therefore, this network is a SOM model with unsupervised learning.

4.4.1 Implementation of the somEF

The following data was supplied to the network: a set of patterns which represent the finite elements in compliance with the codification given in Section 5.2; the initial values for the weights were obtained randomly; and the neighbouring shape was rectangular, in compliance with the description in Section 4.2.
For the training of the SOM networks, an initially high training rate, close to one (1), was used, that should be gradually decreased until a certain acceptable or previously informed value. Up to five hundred (500) iteration cycles were necessary, so that this network could achieve the expected learning rate of the order of 0.01 and equivalent to the one percent (1%) error rate in the results during its execution. This network topology needed a total of fifty thousand (50,000) iteration cycles to reach the proper performance. For this training stage, a sample of fifty percent (50%) of the total elements was used as the set of the network supplied input patterns, to represent the finite elements of the continuous bi-dimensional and tri-dimensional space families shown in Table 7.
After the training, the network should be able to: (a) perform the classification, in terms of equivalent of similar elements (less specialised) over the new element presented to it, according to the equivalence between the already existing patterns; and (b) to present as an answer the class that represents the grouping of the patterns which is identified.

4.4.2 Results of the somEF

During the test performing stages, it was verified that this network was capable of performing many kinds of groupings, according to its own similarity criteria which are based on the characteristics of the self-classification topology. Tables 8 and 9 show some of the groupings this network could perform.
Table 8 shows the classification of the elements according to the last three positions of the input vector *x*, i.e., according to the application, order and element type found, in compliance with the codification described in Section 5.2. This network grouping criteria seems to be related with the identification of patterns which posses

the same values ("1;1;1") on the three last positions and which present different values on the first row. Thus, the grouped patterns were classified as similar patterns by this topology.

Table 8: Grouping of the similar elements by application, order and element type

Elements
P=[2;1;**1;1;1**]
P=[2;2;**1;1;1**]
P=[3;3;**1;1;1**]
P=[3;4;**1;1;1**]
P=[3;5;**1;1;1**]

Table 9 shows the classification of the elements according to the first two positions of the input vector x, i.e., according to the common family and shape found, in conformity with the codification described in Section 5.2. In this table, the classification by similarity between the patterns was identified by the equality of the values on the first two positions of the vector x ("2;1") and by the difference between the other three.

Table 9: Grouping of the similar elements by family and shape

Elements
P=[**2**;**1**;2;2;1]
P=[**2**;**1**;1;2;2]

According to the needs and goals of the classification of the elements present in FEML, this network should be able to classify the characteristics which describe the patterns with better approximation, considering the values present in the five positions of which the input vector *x* is constituted, i.e., the values of the first three positions of the vector should be equivalent and the two last positions could present some more flexibility between the equivalence or approximation of the values, as described in tables 10 and 11. Such characteristics can be observed in figures 3 and 4 which show the finite element hierarchy which was employed.

Table 10: Grouping of the equivalent elements by family, shape, application and order

Elements
P=[**2**;**1**;**2**;**2**;1]
P=[**2**;**1**;**2**;**2**;4]

Table 11: Grouping of the similar elements by shape

Elements
P=[**2**;1;**2**;**2**;**1**]
P=[**2**;2;**2**;**2**;**1**]

Thus, according to the results obtained about the tests performed on this network, it was verified that it was capable of recognizing only forty percent (40%) of the elements presented, based on the similarities between the elements which it had already known and trained. According to the same results, it could be concluded that this network still could not find a suitable level of learning and that the training stages performed were not sufficient to the network to present satisfactory results complying with the needs of this research. Moreover, it

was verified that, if new cycles were performed during the training – even ten more iterations – the results would get even worse, i.e., they tend to decrease until ten percent (10%) of the recognition of the new patterns.
From this limitation, it is clear that the SOM network is indicated for grouping, visualizing and abstraction of known patterns, but that it is not suitable for the recognition of unknown patterns. Thus, for the recognition or classification of patterns to be performed in a more efficient way, it is advisable that network to be projected be used in conjunction with a supervised learning model.

4.5 – Description of the LvqEF Algorithm

The lvqEF program was created with the goal of performing the classification of the elements from which the input data and a set of classes are known, which represent the universe in question, forcing the network to associate classes and patterns which are the most similar to the set of supplied data.
This network is a SOM model with supervised learning.

4.5.1 Implementation of the lvqEF

The following data was supplied to this network: a set of patterns which represent the finite elements in compliance with the codification described in Section 5.2; a set of representative classes elaborated by the network's Supervisor to be transformed into the classes vector, in compliance with the codification described in Table 7; the weights were randomly initialized; one thousand and fifty hundred (1,500) iteration cycles were used, so that the learning rate of the order of 0.01 could be achieved at each training stage; and a total of four thousand and fifty hundred (4,500) iterations, so that the network could achieve an adequate performance.
For the training stage, again a sample of fifty percent (50%) of the total elements was used as the set of the network supplied input patterns, to represent the finite elements of the continuous bi-dimensional and tri-dimensional space families shown in Table 7. The set of supplied classes, also numerically coded in compliance with Table 7, was obtained through the network's Supervisor, i.e., by a human specialist who holds the knowledge about the domain of the problem. The criteria used by the network's Supervisor to choose this representative classes for the grouping of the patterns performed by the network was based on common characteristics about the family, shape, application and order among the classified finite elements, in compliance with the hierarchical model presented in figures 3 and 4 and also associated in accordance with Table 7.

4.5.2 Results of the lvqEF

The tests performed on the lvqEE tool were a one hundred percent (100%) satisfactory. It was verified that this network was capable of performing the classification with greater efficiency in terms of equivalent or similar elements (less specialised), in relation to the new elements to which it was presented and, also, according to the equivalence between the already existing patterns, obtaining the class which represents the groupings which it identified as the answer. Figure 5 synthesises the practical result of this evaluation.

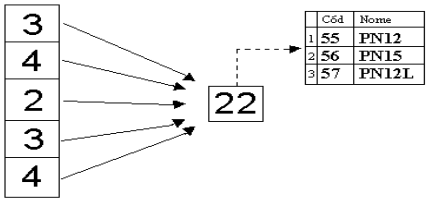

Figure 5: Practical results of the lvqEF

By presenting a new pattern ($[3;4;2;3;4]^T$) to this network, the answer obtained is the number of the class (22) which represents the grouping of patterns that it recognises and which posses characteristics similar to those of the new supplied pattern. According to the semantic descriptions (Code and Name) of the elements from Table 7, the new pattern can be classified as a kind of pattern similar to those grouped and labelled by the class that was found, which testifies the generalisation capacity of the trained model.
In Table 12 a better display of the pattern ($[3;5;3;3;2]^T$), identified by the network, can be seen, presented as an answer the number of the class (30) which represents the grouping of the patterns which it knows and that posses characteristics similar to those of the supplied pattern. According to the descriptions (Code and Name) of the elements from Table 7, it is possible to classify the pattern recognised by the network with precision.

Table 12: Groupings of the equivalent elements

Supplied Pattern	Network Answer (class)	*Classification*	
		Code	*Name*
$[3;5;3;3;2]^T$	30	71	HF20

5. FINAL CONSIDERATIONS

The problem of data transportation between different formats and platforms requires the constant application of new technologies, processing agility and trustworthiness on the information transport. These are plausible arguments to the appearance of new proposals of research. In this sense, there is still much to do beyond what has already been done until the present days.
As it was displayed in this present work, the classEF network was not capable of classifying the unknown patterns (new patterns) in accordance to the similarities with the patterns that the network knows. The results prove that this kind of network (MLP) has better applicability in classification when the information about the problem's domains and its relations are previously owned, not showing itself as applicable to the solution of the problem in question.
The somEF network has poorly recognized the elements to which it was presented according to their equivalence with the elements which it has trained and has been previously exposed to, and, therefore, it is inadequate according to FEML necessities. However, the capacity of this type of network to perform many kinds of groupings, which emphasises its applicability in problems which require visualisation and abstraction of the data between a set of known patterns, was certified.
The results and the analyses performed state that the lvqEF network is the most applicable among the topologies presented here. It was capable of efficiently recognise groupings of similar or equivalent patterns in a same class, allowing for the inclusion of new patterns and, therefore, readily taking care of the necessities and the classification goals of the finite elements pointed by the FEML.
It is expected, as a result, that this tool can contribute, giving greater applicability to the FEML language as an effective data interchange standard for FEM programs, allowing to the interface programs of the finite element software to dynamically optimise the importation process between the proprietary and the neutral formats.

6. REFERENCES

[1] Moita GF, Pinheiro MC. Towards a standard for finite element data exchange using XML. CD-Rom Conference Proceedings of the 7th U.S. National Congress on Computational Mechanics, Albuquerque, USA, 2003.

[2] Zurada JM, Introduction to Artificial Neural Systems. Boston, MA: PWS Publishing Company, 1995.

[3] Pinheiro MC. A Proposition of a XML Schema for the Interchange of data between Finite Element Programs, M.Sc. Dissertation, Program in Technology, CEFET-MG, Belo Horizonte, Brazil, 2003 (in Portuguese).

[4] Connolly D, Khare R, Rifkin A. The evolution of Web documents: the ascent of XML. World Wide Web Journal 1997; 2 (4): 119-128.

[5] Marchal B, XML by example. 1ª Edition. Indianapolis, EUA, 2000.

[6] W3C – World Wide Web Consortium. A Conversion Tool from DTD to XML Schema. 2000. Available in <http://www.w3.org/2000/04/ schema_hack>. Access: Mar 2006.

[7] Lee D, Chu WW. Comparative Analysis of Six Schema Languages. ACM Sigmod Record 2000; 29 (3).

[8] LUSAS. *User Guide*. FEA Ltd. United Kingdom, 2004.

[9] Meireles MRG, Almeida PEM, Simões MG. A Comprehensive Review About Industrial Applicability of Artificial Neural Networks. IEEE Trans. on Industrial Electronics 2003; 50 (3): 585-601.

[10] Oliveira DMLM. An Intelligent Hierarchical Classification of Finite Elements within a XML Schema, M.Sc. Dissertation, Program in Mathematical and Computational Modeling, CEFET-MG, Belo Horizonte, Brazil, 2005 (in Portuguese).

[11] Oliveira DMLM, Pinheiro MC, Moita GF. On the Development of a Neutral Format with Intelligent Element Classification for Data Sharing Among FE Programs. CD Rom Conference Proceedings of ECT 2006 - The Fifth International Conference on Engineering Computational Technique, B.H.V. Topping, (Editor), Civil-Comp Press, Stirling, United Kingdom, 2006.

[12] MATLAB. *MathWorks Products Used*. Available in: <www.mathworks.com>. Access: Sept. 2008.