Computing and Informatics, Vol. 34, 2015, 425–457

PETRI NETS MODELING OF DEAD-END REFINEMENT PROBLEMS IN A 3D ANISOTROPIC *HP*-ADAPTIVE FINITE ELEMENT METHOD

Arkadiusz Szymczak, Maciej Paszyński

AGH University of Science and Technology, Krakow, Poland
e-mail: arek.szymczak@gmail.com, maciej.Paszynski@agh.edu.pl

David Pardo

Department of Applied Mathematics, Statistics, and Operational Research at the University of the Basque Country (UPV/EHU), Bilbao, Spain
Basque Center for Applied Mathematics (BCAM), Bilbao, Spain
Basque Foundation for Science (IKERBASQUE), Bilbao, Spain
e-mail: dzubiaur@gmail.com

Anna Paszyńska

Jagiellonian University, Krakow, Poland
e-mail: anna.Paszynska@uj.edu.pl

Abstract. We consider two graph grammar based Petri nets models for anisotropic refinements of three dimensional hexahedral grids. The first one detects possible dead-end problems during the graph grammar based anisotropic refinements of the mesh. The second one employs an enhanced graph grammar model that is actually dead-end free. We apply the resulting algorithm to the simulation of resistivity logging measurements for estimating the location of underground oil and/or gas formations. The graph grammar based Petri net models allow to fix the self-adaptive mesh refinement algorithm and finish the adaptive computations with the required accuracy needed by the numerical solution.

Keywords: Petri nets, automatic *hp* adaptivity, finite element method, dead-end, borehole resistivity logging

Mathematics Subject Classification 2010: 68Q05, 68Q42, 68Q60, 68Q85

1 INTRODUCTION

Isotropic mesh refinements break selected finite elements into three directions to construct eight son elements. Anisotropic refinements break selected elements into one, two or three directions, producing two, four or eight element sons, respectively. We consider anisotropic mesh refinement algorithms for 3D grids composed with hexahedral finite elements, as presented in [9]. Improper implementation of the anisotropic *h*-adaptation algorithm may result in a dead-end scenario, where some requested refinements are impossible to execute. To make the implementation of local refinements tractable while ensuring continuity in the finite element solution, many anisotropic refinement codes support the so-called *1-irregularity rule*. According to that rule an element with hanging nodes cannot be further refined. For refining such an element one needs at first to refine one or several neighboring elements in order to eliminate all hanging nodes. The problem is illustrated in Figures 1 and 2. Let us consider two finite elements, one broken into eight son elements, and the other one unbroken. In the finite element method nomenclature the nodes and vertexes of the shared face are called constrained in the sense that the approximation over four small faces of the small elements is constrained by the approximation over one big face of the big element.

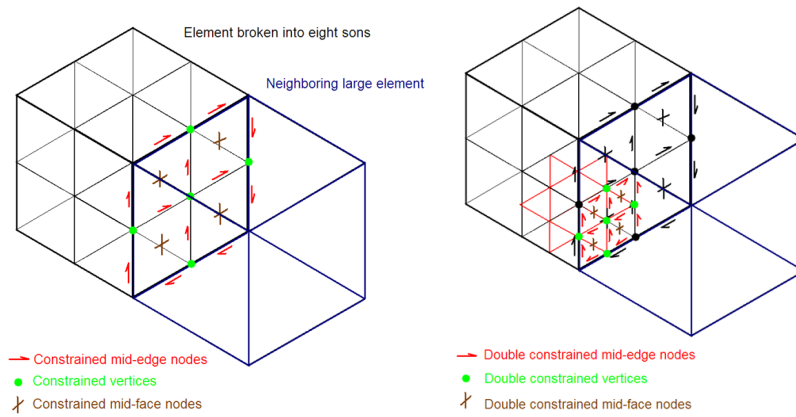

Figure 1. Left panel: Two adjacent elements, one broken into eight son elements. Right panel: The forbidden state with double constrained nodes.

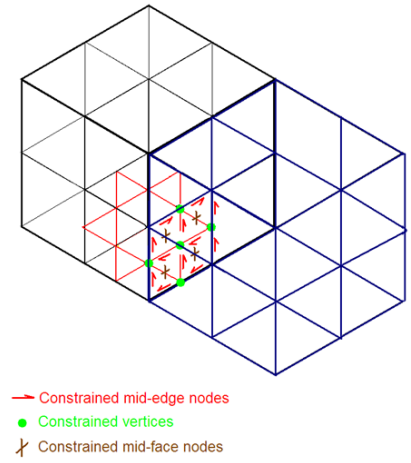

Figure 2. The breaking of large element followed by breaking of one of the small elements

If we break one of the small elements for the second time, the resulting state is forbidden, compare left panel in Figure 1. The reason is that in such a case, we will have double constrained nodes over the smallest faces of the small broken element. To prevent such forbidden state, it is necessary to break the large neighbor before breaking the small element, compare Figure 2. As it comes from numerical experiments, the 1-irregularity rule has one unexpected drawback, both in two and three dimensions. Namely, it may result in a dead-end of the adaptation process. It should be emphasized that the mesh at dead-end state is at the acceptable state, the numerical problem can be solved on that mesh, however further refinements are not possible here. The practical motivation of this paper was the personal communication with David Pardo, working on 3D anisotropic mesh refinements algorithm used for the simulations of 3D DC resistivity logging measurements in deviated wells [23, 20, 24, 15]. During these computations the dead-end problem occurred. This 1-irregularity rule implies that an edge of an element can be adjacent to no more than two smaller edges. Additionally, a face of an element can be adjacent to either two smaller faces, or to four smaller faces, provided they are broken in both directions. In the adaptive community, it is often said that the small edge is constrained by the big edge, or the small face is constrained by the big face. One of the first dead-end problems for 3D h-adaptive computations with hexahedral elements was identified in [10]. The dead-end in that version of the adaptive code was caused by the fact that elements could be broken only into two son elements, either along **X** or **Y** or **Z** direction. In that paper [10] Figure 13 (reproduced here on left panel in Figure 3 with authors' agreement) illustrates a basic dead-end scenario. There are three elements in a row, the first one and the third one are broken in two different directions. Another request to break the first element implies the necessity of breaking the central element. This is because

the left side face of the central element cannot be adjacent to two times broken faces of the first element. If we break the central element in two directions, we will block the possibility of breaking the third element, since the central element must be broken in another direction in such case. This dead-end was overcome by adding the possibility of breaking elements into four or eight son elements at the same time [10].

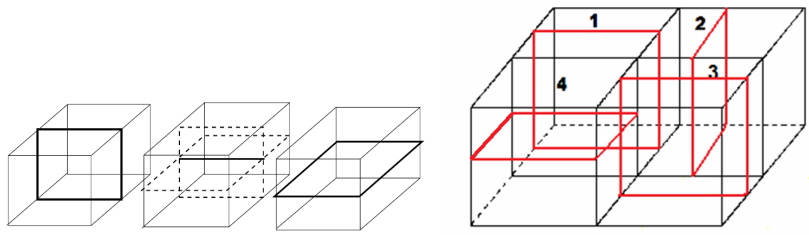

Figure 3. Left panel: A first dead-end scenario. Right panel: A second dead-end scenario.

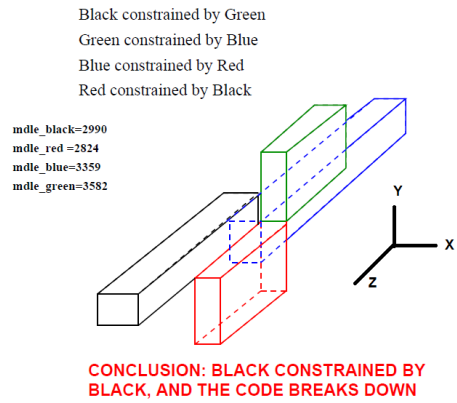

Figure 4. A third dead-end scenario

Actually, the authors of [10] found out that this additional breaking of elements should be performed into eight son nodes, to block the unwanted propagation of additional refinements. The authors of this paper encountered another dead-end scenario, presented on right panel in Figure 3. In this case, we have a patch of four elements. Each of these elements have been broken into two son elements. The resulting mesh does not violate the 1-irregularity rule, however, further refinements are not possible in this patch. For example, let us assume that we want to break element 1 again into the direction perpendicular to the **X** axis. The right face of element 1 is constrained by the left face of element 2. We need to break element 2 first, into eight son nodes, in order to prevent further propagation of refinements.

But we cannot do that, since the front face of element 2 is constrained by the rear face of element 3. We need to break element 3 into eight son nodes, but we cannot do that since the left face of element 3 is constrained by the right face of element 4. So we need to break element 4 into eight son nodes, but we cannot do that yet, since the rear face of element 4 is constrained by the front face of element 1. We have to break element 1 first, in the way that is contradictory with its original request for refinement. We have a dead-end scenario here. Later, we found another dead-end problem, much more complicated. In the 3D mesh described in Figure 4, there are four elements that touch each other through edges. We would like to break the black element one more time into four son nodes, along **Z** axis. This refinement request implies the necessity of breaking the green element, since the edge of the black element perpendicular to **Y** axis is constrained by the edge of the green element perpendicular to **Y** axis. We want to break green element into eight son nodes, to prevent unwanted propagation of refinements. But this is not possible yet, since the **Z** edge of the green element is constrained by the **Z** edge of the blue element. The blue element also must be broken into eight son elements. But again, this is not possible, since **Y** edge of the blue element is constrained by **Y** edge of the red element. In turn, the Z edge of the red element is constrained by the **Z** edge of the black element. Again, we encounter a dead-end scenario here. Other dead-end scenarios have been also reported in non-structural 3D tetrahedral adaptive finite element method computations [18]. In this paper we propose the use of a Petri net for detecting a dead-end scenario. The modeling of a dead-end scenario for adaptive finite element methods was already performed for two dimensional (2D) anisotropic refinements of rectangular meshes. In the first attempt [40], we modeled only a 2D sub-mesh with 2×2 rectangular elements, and considered a Petri net modeling a dead-end scenario in such simple example. This result was generalized for arbitrary 2D rectangular grids in [41]. Later, in [32] we designed a Petri net model for a 3D sub-mesh with $2 \times 2 \times 2$ hexahedral elements, and showed how to remove the dead-end in such example. In this paper, we generalize these results to the class of arbitrary 3D hexahedral meshes in a similar way as it was performed in [41] for the case of 2D rectangular grids [40]. The Petri net model is independent of the numerical problem being solved, however it depends on the particular implementation of the mesh adaptation algorithm. The Petri net model has been implemented in PIPE software [7], the reachability graph has been constructed there, and the dead-end analysis has been executed by using automatic tools implemented in PIPE software. Once we have a corrected version of the adaptation algorithm proven to be dead-end free, this algorithm can be used to solve any numerical problem without incurring in a dead-end scenario. The computational mesh can be represented as a graph and mesh refinements can be modeled as graph grammar productions (graph transformations),in both 2D [30, 31] and 3D [27, 28]. Thus, in this paper we introduce a formal model of a graph representing a 3D computational mesh with hexahedral elements, subject to anisotropic refinements. We also provide formal definitions of mesh transformations expressed as graph grammar productions, representing anisotropic mesh refinements. The graph grammar model definitions

presented in this paper are a generalization of the one presented in [27, 28] for the isotropic mesh refinements. The paper is organized in the following way. Section 2 provides an overview of sequential adaptivity mesh refinement codes. In Section 3 we introduce formal definitions of Composite-Programmable graph (CP-graph) and Composite Programmable graph grammar (CP-graph grammar) models expressing the h adaptation process. In Section 4, we introduce a graph grammar model describing adaptive mesh refinements. Next, in Section 5 we introduce a mesh adaptation algorithm as implemented in the *hp3d* code in [26]. Section 6 presents the algorithm for generation of hierarchical Petri nets modeling the mesh refinements algorithm implemented in [26]. We show that this Petri net model does not prevent possible dead-end scenarios. Section 7 describes the enhanced graph transformations, the corrected mesh adaptation algorithm, the Petri net model expressing the new algorithm and the proof that the new Petri net model is dead-end free. The paper is concluded in Section 8 with numerical results concerning the anisotropic mesh refinements algorithm used for simulations of resistivity logging measurements in deviated wells.

2 RELATED WORKS

The adaptive algorithms can be classified in the following way:

- *Uniform h adaptation*: all finite elements are uniformly broken into smaller elements.
- *Uniform p adaptation*: the polynomial order of approximation is increased uniformly over the entire mesh, e.g. by adding bubble shape functions of the higher orders over element edges and interiors.
- *Non-uniform h adaptation*: some finite elements are broken into smaller elements, only in those parts of the mesh which have a high numerical error.
- *Non-uniform hp adaptation*: some finite elements are broken into smaller elements, and the polynomial orders of approximation are increased, only in those parts of the mesh which have a high numerical error.
- *r adaptation*, where the mesh is re-generated using new distribution of elements.

For non-uniform h or hp adaptation, it is necessary to locate finite elements with a high numerical error and select the optimal refinements for them. The non-uniform h or hp adaptation process can be executed in the following ways:

- The selection of the finite elements to be refined and the type of refinement depends on a user.
- The selection of the finite elements to be refined and the type of refinement depends on an algorithm based on the knowledge of the structure of the solution.

- The selection of the finite elements to be refined and the type of refinement depends on the self-adaptive algorithm, which is designed without any particular knowledge of the structure of the solution, and works in a fully automatic mode, without any user's interaction.

The first and the second algorithm are referred to as the non-automatic adaptation, whereas the third algorithm is called automatic adaptation. In particular, the non-uniform *hp* automatic adaptation is called the self-adaptive *hp* Finite Element Method (self-adaptive *hp*-FEM). The *r* adaptation is also often referred to as *re-meshing*. Algorithms for the uniform *h*, uniform *p*, non-uniform *h* and non-uniform *hp* automatic adaptation for 3D grids have been designed, implemented and tested by the group of prof. Leszek Demkowicz [9]. Many authors followed the approach originated by Demkowicz and implemented their own variations of these algorithms. In [21], authors employ modern *h* and *hp* adaptation algorithms for the Girkmann problem. [2] presents the *h* adaptation approach using the octree data structure and the ideas originally introduced by [9] for local *h* refinements. The uniform *h* adaptation algorithm has also been utilized for the solution of the projection problem [16]. The *r*-adaptation is also commonly used in the computational community. Paper [17] uses the re-meshing algorithm for modeling large deformations in geological problems. The *r* adaptation algorithm can also be utilized for solution of non-stationary problems, e.g. wind flow around the bridge [36], or flow problem [4]. The self-adaptive *h*-FEM or *hp*-FEM algorithm may utilize different error estimators for guiding the adaptation process. There are different error estimators defined for elliptic [3, 5], parabolic [14, 6] or multi-physics problems [22]. From the point of view of the dead-end modeling, the error estimator does not influence the problem. In this paper, we focus on modeling dead-end scenarios in the self-adaptive *h*-FEM algorithm. The addition of automatic *p* adaptivity is also possible, since playing with different polynomial orders of approximation over the edge does not influence the dead-end problem, which results from *h* adaptation only. The Petri net model presented in this paper can be also applied to model the other non-automatic adaptation algorithms. The only exception is the *r* adaptivity.

The parallelization of any of the above adaptation algorithm results in distribution of the computational mesh into processors. The Sierra Environment is a platform supporting *h* refinements over the mesh distributed into subdomains [13]. There is also object-oriented distributed data structure *hp*-adaptive flow simulation code [1]. The original *hp*-adaptive algorithm from [9] has also been parallelized using either domain decomposition approach [33] or OpenMP approach [37]. The other parallel *hp* adaptive algorithms implemented so far have been developed by [12, 38] in context of Discontinuous Galerkin DG) methods. The parallel *hp* adaptive algorithms for Continuous Galerkin have been developed by [35, 19].

Our model can still be applied for analyzing the dead-end problem for parallel mesh refinements algorithm, assuming the dead-end scenario takes place over a single sub-domain, or the domains have been collected into a single processor. There are some alternative non-adaptive efficient parallel methods for solution of compu-

tationally intensive problems, like the alternative direction solver described in [34], but they do not generate deadlock problem since they do not use adaptive grids. The adaptive solvers can be also used for solution of some inverse problems, and the parallelization there may concern different calls to the solver performed at the same time [29], not necessary using the mesh partitioning methods.

3 GRAPH GRAMMAR MODEL OF MESH TRANSFORMATIONS

Definition 1. The Composite Programmable Graph (CP-graph) modeling a mesh with hexahedral elements is defined as

$$CP_{mesh} = (V, E, \xi_V, P, P1_E, P2_E, P1_F, \ldots, P4_F, P1_I, \ldots, P8_I, virt\_ref) . \quad (1)$$

- V is a set of nodes.
- E is a set of edges, such that $E \subseteq B(V) \times B(V)$ fulfills the following conditions:
  - for each $(i,u) \in B(V)$, there exists at most one $(j,v) \in B(V)$ such that $((j,v),(i,u)) \in E$,
  - for each $((j,v),(i,u)) \in E$, $v \neq u$.
- For a set V of nodes and a node labeling function $\xi_V : V \mapsto W$, let $B(V) = \bigcup_{v \in V} \beta(\xi_V(v)) \times \{v\}$ denotes the set of pairs (i,v) called *bonds*, where $i \in \beta(\xi_V(v))$ and $v \in V$.
- Natural numbers i are called indexes of bonds.
- Let W be a finite, nonempty subset of $A_V \times [i]_N$, where α and β are the projections of each w in W to the first and the second component, respectively, i.e.,
$$w = (\alpha(w), \beta(w)), w \in W. \quad (2)$$
- The set of $W \subset A_V \times [i]_N$ is called the set of *extended labels* over A_V and $[i]_N$.

i denotes the interval $1, \ldots, i$ for $i \geq 0$ (with $[0] = \emptyset$). $[i]_N$ denotes a family of intervals [i] for $i \geq 0$.

- A_V is an alphabet of node labels
$$A_V = A_V^1 \cup A_V^2 \cup A_V^3 \cup A_V^4 \quad (3)$$
  - $A_V^1 = \{v\}$ is a set of node labels that denote vertexes of finite elements
  - $A_V^2 = \{F\}$ is a set of node labels that denote edges of finite elements
  - $A_V^3 = \{E\}$ is a set of node labels that denote faces of finite elements
  - $A_V^4 = \{I, i\}$ is a set of node labels that denote interiors of finite elements.
- $P : V \times A_V^1 \mapsto \{R \times R \times R\}$ is a function attributing nodes, which assigns the coordinates to each vertex of the element.

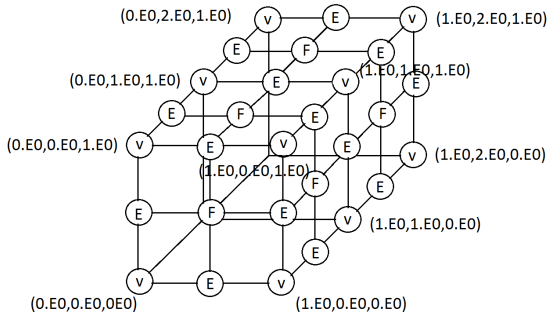

Figure 5. CP-graph representation of the two finite element mesh

The exemplary CP-graph representation of two element mesh is presented in Figure 5.

Definition 2. Let $CP_{mesh} = (V, E, \xi_V, P, P1_E, P2_E, P1_F, \ldots, P4_F, P1_I, \ldots, P8_I, virt\_ref)$ and $CP'_{mesh} = (V', E', \xi'_V, P', P1'_E, P2'_E, P1'_F, \ldots, P4'_F, P1'_I, \ldots, P8'_I, virt\_ref')$ be two CP-graphs over W, as defined in Definition 1. Graph CP_{mesh} is isomorphic with graph CP'_{mesh} if and only if there exists a bijection $f : V \to V'$ such that

- $\bigvee u, v \in V, e = ((i, v), (j, u)) \in E$ iff $e' = ((i, f(v)), (j, f(u))) \in E'$
- $\forall v \in V, \xi_V(v) = \xi_{V'}(f(v))$
- $\forall v \in V, attr(v, \xi_V(v)) = attr'(f(v), \xi_{V'}(f(v)))$
  where $attr \in \{P, P1_E, P2_E, P1_F, \ldots, P4_F, P1_I, \ldots, P8_I, virt\_ref\}$
  and $attr' \in \{P', P1'_E, P2'_E, P1'_F, \ldots, P4'_F, P1'_I, \ldots, P8'_I, virt\_ref'\}$

Definition 3. A Composite Programmable graph grammar (CP-graph grammar) G for mesh adaptation is defined as:

$$G = (W, \xi_V, V, GGP, x) \tag{4}$$

where:

- V is a set of nodes
- W is a set of extended labels
- ξ_V is a node labeling function
- GGP is a finite set of pairs of (l, r) called *graph grammar productions*, where l and r are composite graphs over W of the same type, and the set of free bonds of r and l are equipped with ordering relations
- x is a labeled node from V and it is called the **axiom** of the grammar.

Let $p = (l, r) \in GGP$ be a production in G. The first l and second r element of the pair are called the *left-hand-side* and *right-hand-side* of p. The application of production p to a composite graph c consists of substituting the composite graph r by a subgraph of the graph c isomorphic to the composite graph l and replacing the connections of l with the connections of r in such a way that each free bond of l is substituted by a free bond of r with the same order number.

4 GRAPH GRAMMAR PRODUCTIONS FOR MESH ADAPTATION

In this section, we define the CP-graph grammar model of the mesh refinement algorithm from [9]. The algorithm has been implemented in *hp3d* code. The CP-graph grammar is introduced here by defining the graph grammar productions expressing the mesh transformation rules. The productions use graphical transformation of sub-graphs of the CP-graph representing the computational mesh being refined. Some representative graph grammar productions from the first set of productions are presented in Figures 6–8. Productions whose names start with **V** denote the so-called virtual refinements requests for breaking an element interior in one of several possible directions. Productions **(VX)**, **(VY)** and **(VZ)** denote virtual breaking of an element interior in a single direction along X, Y and Z, axis, respectively. Productions **(VXY)**, **(VYZ)** and **(VXZ)** denote virtual breaking of an element interior along two designated axis at the same time. Production **(VXYZ)** denotes virtual breaking of element interior along all three axis at the same time.

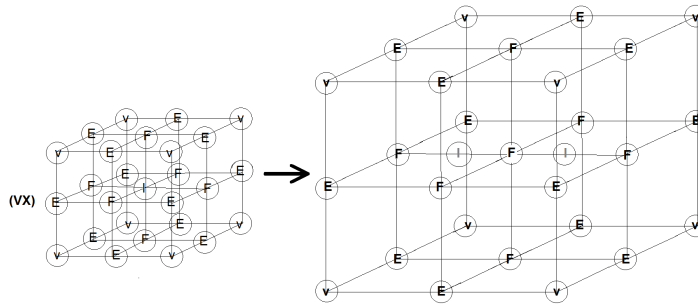

Figure 6. Graph grammar production **(VX)** for virtual refinement of a single element into X direction

The computational mesh after execution of any of these virtual refinements is not in the legal state. This is because the virtual refinements break only element interiors, and the mesh must be closed. This is done by enforcing additional breaking of some faces and edges. A face must be broken if it is surrounded by two broken interiors. An edge can be broken if it is surrounded by four broken faces. The execution of virtual refinements is followed by the execution of several graph grammar productions, checking the connectivities between edges, faces and interiors, and

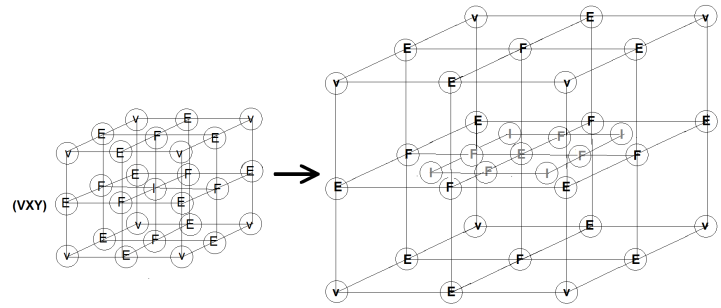

Figure 7. Graph grammar production **(VXY)** for virtual refinement of a single element into X and Y directions

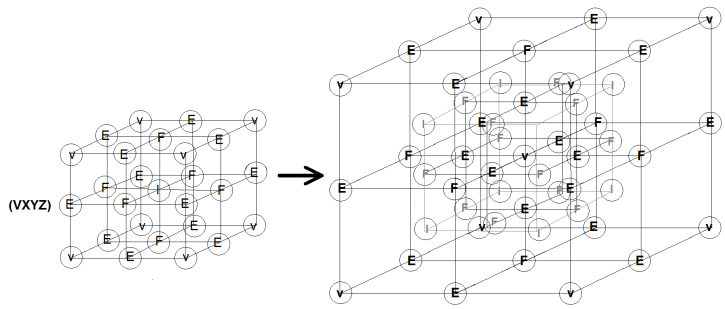

Figure 8. Graph grammar production **(VXYZ)** for virtual refinement of a single element into X, Y and Z directions

enforcing some additional refinements. The graph grammar productions for breaking faces surrounded by broken interiors are presented in Figures 9–11. Production **(PBF1)** presented in Figure 9 describes the process of breaking a face surrounded by one interior that has already been broken in one direction. There are analogous productions **(PBF1_11)** and **(PBF11)** presented in Figures 10 and 11 which describe the process of breaking a face surrounded by one interior that has already been broken in one direction and another interior that has already been broken in two directions, and the process of breaking a face surrounded by two interiors that have already been broken in two directions.

The exemplary representative graph grammar production for breaking edges surrounded by broken faces is presented in Figure 12. There are similar three graph grammar productions, since an edge may be surrounded by two, three or four faces.

For each element with broken interior (after the virtual refinement) we execute the graph grammar productions **(PBF1)**, **(PBF1_11)**, **(PBF11)** and **(PEB2)**, **(PEB3)**, **(PEB4)** for its faces and edges. The distinction between virtual and

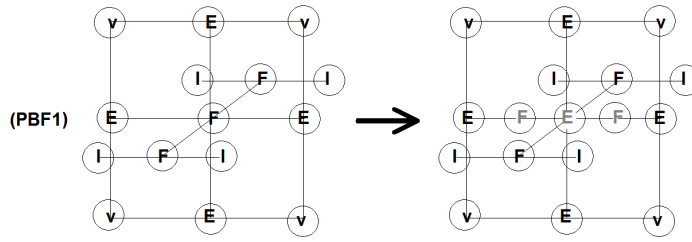

Figure 9. Graph grammar production **(PBF1)** enforcing breaking of a face surrounded by two interiors broken in one direction

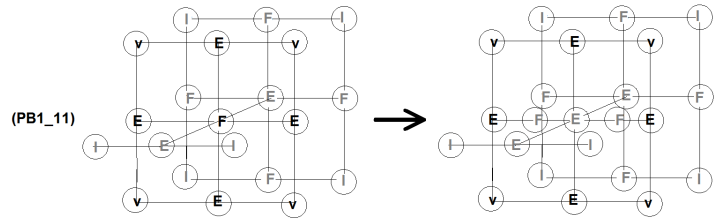

Figure 10. Graph grammar production **(PBF1_11)** enforcing breaking of a face surrounded by one interior broken in one direction and another interior broken in two directions

actual refinements is needed for the dead-end detection during the refinement propagation. Both the productions for virtual and actual refinements are quite complex when they are expressed in a formal way using CP-graph grammar notation. Thus, in the following sections we will use the simplified notation summarized in Figure 13. In this simplified notation, we assume that closing of the refinement process for faces and edges can be expressed by one production **B*** whose name corresponds to the virtual refinement executed before. For example, execution of production

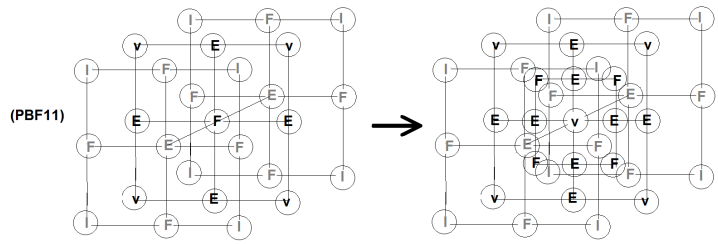

Figure 11. Graph grammar production **(PBF11)** enforcing breaking of a face surrounded by two interiors broken in two directions

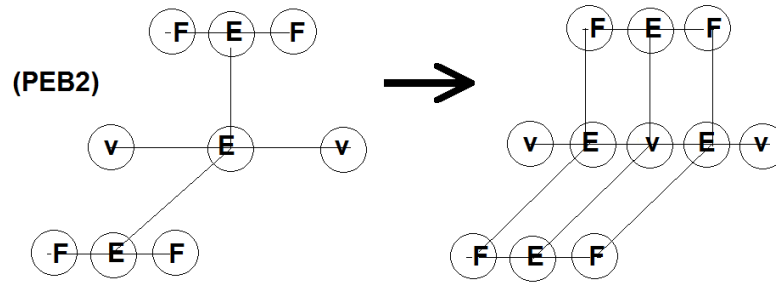

Figure 12. Graph grammar production **(PEB2)** enforcing breaking of an edge surrounded by two broken faces

(BY) after the production **(VY)** requires execution of productions **(PBF1) and (PBF1_11)** for all the faces of the element surrounded by broken interiors, as well as execution of production **(PEB2)** for all the edges of the element surrounded by broken faces.

5 MESH ADAPTATION ALGORITHM

The mesh adaptation algorithm as implemented in *hp3d* code [9], can be summarized in the following way:

Algorithm 1.

```
1   L = List of elements el to be refined with refinement kind
2   do while  L not empty
3     el = get next element from L and its refinement kind
4     loop through face ∈ faces of element el
5       if face belongs to big neighbor element then
6         Store neighbor with its refinement kind at the end of L
7         Store el and its refinement kind at the end of L
8         continue do-loop from line 2
9       endif
10    enddo
11    loop through edge ∈ edges of element el
12      if edge belongs to big neighbor element then
13        Store neighbor with its refinement kind at the end of L
14        Store el and its refinement kind at the end of L
15        continue do-loop from line 2
16      endif
17    enddo
18    break element el in a way kind using the virtual refinement
19  end while
```

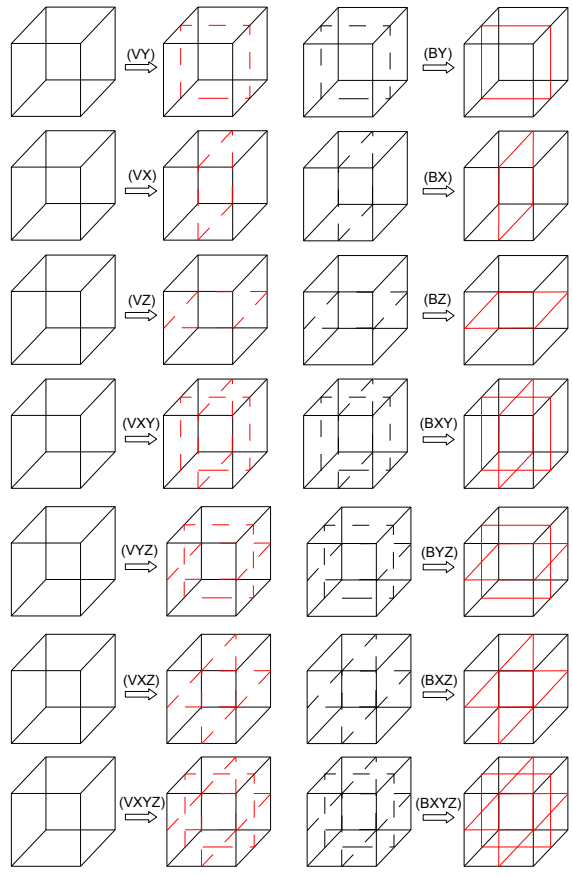

Figure 13. Graph grammar production for anisotropic breaking of a single element

Remark 1. A **dead-end** scenario occurs when a virtual refinement propagates into some adjacent element either through a face or an edge, and it is contradictory to the virtual refinement that has already been selected for the adjacent element.

In the mesh adaptation Algorithm 1, the dead-end happens in line 18, where we try to break the interior of an element that has already been broken in some other way.

6 HIERARCHICAL PETRI NET MODEL

Remark 2. In order to detect a possible dead-end scenario, we need to construct a hierarchical Petri net with the main page covering the entire mesh, and sub-pages corresponding to pairs of elements either through a face or through an edge.

We construct the hierarchical Petri net model for the entire mesh, with main page corresponding to the entire mesh, and with sub-pages corresponding to all element pairs, either through a face or through an edge. Because of the 1-irregularity rule enforced over the entire mesh, all the pairs of elements are at the same level of adaptation. Each sub-page corresponding to a single pair of elements considers refinement request for any of the elements in the pair, with possible propagation to the other element from the pair. The sub-page considers also all possible refinement requests coming from the external elements. Thus, we consider all possible combinations of two virtual refinement requests, and check if they result in a dead-end. The hierarchical Petri net model has been constructed in such a way that actual dead-end detection is performed by the sub-pages covering two-element patches of the mesh. The hierarchical Petri net sub-page for finite elements adjacent along X axis is depicted in Figures 14 and 15. For the sake of saving space, we skip similar hierarchical Petri net sub-pages for finite elements adjacent along Y and Z directions. The Petri nets considered in this paper are defined as hierarchical colored Petri nets (compare [39], p. 177, definition 10.16) where we utilize only one color (we have only one type of token). We use the composition method like for the hierarchical colored Petri net with fusion of places. The sub-pages are related with the main page by socket and port nodes. In particular, the socket is the place in the main page of the hierarchical Petri net that is shared with the sub-page. The socket place from the point of view of a sub-page is called a port. In other words, the socket and port are the two names of the same place, common for main page and for some sub-page (compare [39], p. 176).

Since the propagation of dead-end may also occur through an edge, as it is depicted in Figure 4, it is also necessary to consider patches of elements adjacent through edges. Each element has six neighbors through faces, where there are actually three symmetric Petri nets, for adjacency along X, Y and Z directions. There are also twelve edges, and there are twelve possible adjacent neighbors through edges. The Petri net sub-page for adjacency through edges is similar to the sub-pages reflecting the adjacency through faces, but only the refinements in the direction perpendicular to the edge may occur there. For sake of saving space, we do not present this Petri net here. The Petri net arcs define all possible execution paths for a round of mesh element refinements by enforcing dependency relationships between relevant transitions (productions). Whenever only one of a set of grammar productions can be executed, the corresponding Petri net transitions depend on a common place with a single token. Whenever execution of a production blocks execution of another production, an inhibitor arc is used between the corresponding Petri net transitions (actually between the intermediate place and the dependent transition).

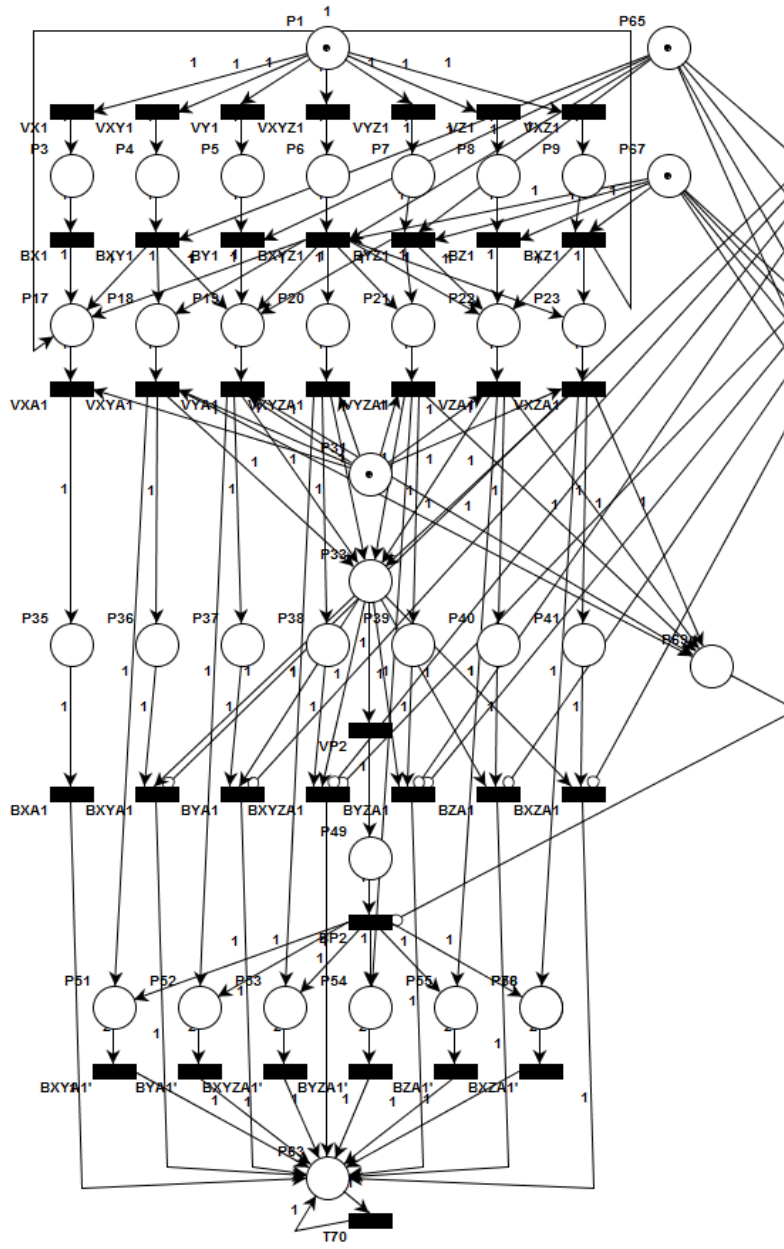

Figure 14. First part of the hierarchical Petri net with a dead-end for finite elements adjacent along X axis

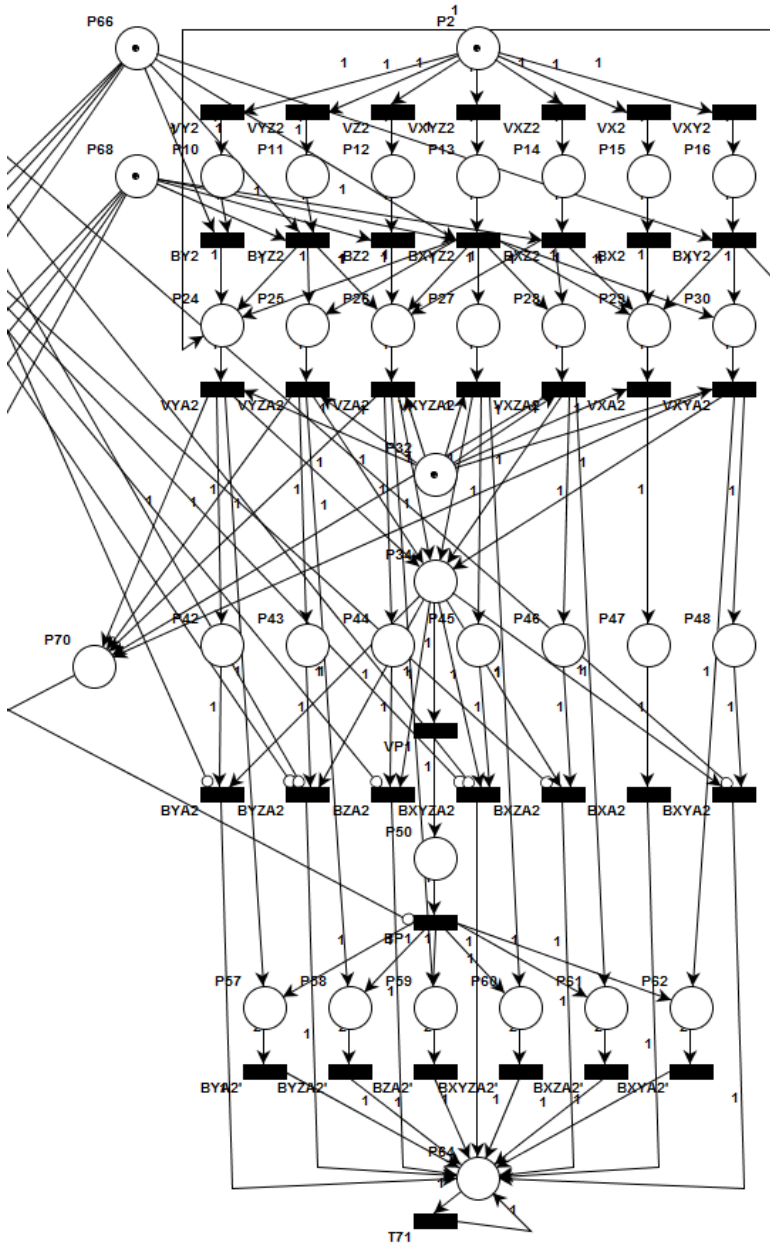

Figure 15. Second part of the hierarchical Petri net with a dead-end for finite elements adjacent along X axis

Each hierarchical Petri net subpage contains two starting places (**P1** and **P2**) – one for each mesh element of the modeled pair. **P1** and **P2** are fusion places shared between all sub-pages covering common mesh elements (a given mesh element can be part of up to six element pairs). The two upper rows of Petri net transitions are named after the grammar productions they represent. Numbers at the end of transition names denote the corresponding mesh element to which a given transition pertain to. Firing any of those transitions models executes a corresponding grammar production. The remaining Petri net transitions model the following mesh element transformations:

- **VXA** – request (virtual) to break along X axis the adjacent sub-element
- **VYA** – request (virtual) to break along Y axis the adjacent sub-element
- **VZA** – request (virtual) to break along Z axis the adjacent sub-element
- **VXYA** – request (virtual) to break along X and Y axis the adjacent sub-element
- **VXZA** – request (virtual) to break along X and Z axis the adjacent sub-element
- **VYZA** – request (virtual) to break along Y and Z axis the adjacent sub-element
- **BXA** – transformation breaking along X axis the adjacent sub-element
- **BYA** – transformation breaking along Y axis the adjacent sub-element
- **BZA** – transformation breaking along Z axis the adjacent sub-element
- **BXYA** – transformation breaking along X and Y axis the adjacent sub-element
- **BXZA** – transformation breaking along X and Z axis the adjacent sub-element
- **BYZA** – transformation breaking along Y and Z axis the adjacent sub-element
- **BXYZA** – transformation breaking along all 3 axis the adjacent sub-element
- **VP** – propagation of (virtual) refinement request onto the other element
- **BP** – transformation executing the propagated refinement
- **VB** – transformation converting any virtual refinement into a three-directional virtual refinement.

Transitions whose names end with prim model the same graph transformations as the corresponding transitions without prim at the end of the name but reachable by a different execution path (that is, with vs. without refinement propagation). A mesh (sub-)element can be broken for the second time (transitions **B** [**D**]**A**[#], where [**D**] stands for any direction(s) and [#] is the number of concerned mesh element) only when the adjacent element is already broken in the same direction at least once. This can be achieved in either of the following two ways:

1. The adjacent element has been broken in the required direction independently.
2. The required refinement is propagated onto the adjacent element (e.g. **P33** $\rightarrow$ **VP2** $\rightarrow$ **P49** $\rightarrow$ **BP2** for the "left" element in the pair).

An alternative to the above two scenarios is modeled in the Petri net by means of places **P65**–**P70**. Single breaking of a mesh element in a set **D** of directions uninhibits the single breaking of the adjacent (sub-)element in a subset **D** of directions (places **P65**–**P68**). A second virtual refinement of a mesh element in fewer than three directions at the same time inhibits the refinement to be propagated from the adjacent mesh element (places **P69** and **P70**). It is critical that each pair of adjacent mesh elements is covered with a Petri net subpage of appropriate type. To this end, the hierarchical Petri net generation algorithm for a given finite element mesh has been developed.

Assumption 1. All elements of the mesh to be analyzed are at the same adaptation level. This is a direct consequence of the 1-irregularity rule that must be fulfilled over the mesh.

Algorithm 2. Generation of a hierarchical Petri net for a given 3D finite element mesh

1. Create the main page of the hierarchical Petri net.
   - Create a Petri net place for each element of the mesh being analyzed.
   - For each pair of adjacent (by face in any direction: along X, Y or Z axis; or by any of the twelve edges) mesh elements, create a Petri net transition and connect the corresponding places to this transition with arcs.
   - Create two output places for each transition and connect each place to its corresponding transition with an arc.
2. Bind the main page of the hierarchical Petri net with the sub-pages.
   - Substitute each transition in the main page with an instance of appropriate sub-page type, depending on whether the input places to the given transition represent mesh elements that are face-adjacent along X, Y or Z axis, or adjacent by one of the twelve edges.
   - The input places of each transition in the main page become the socket nodes to the substituted sub-page instance and are bound to the port nodes (places **P1** and **P2**) in the substituted sub-page instance.
3. The output places of each transition in the main page become the socket nodes to the substituted sub-page instance and are bound to the port nodes (places **P63** and **P64**) in the substituted sub-page instance.
4. Each port node is a global fusion, *i.e.* there is a single instance of given place shared by all sub-page instances of the hierarchical Petri net.

Figure 17 presents the main page of hierarchical Petri net generated for an exemplary computational mesh consisting of 8 elements, depicted in Figure 16. Places **Elem**[#] correspond to mesh elements with a given number. All **Elem**[#] places in the main page are input sockets, bound to port nodes (places **P1** and **P2**) of

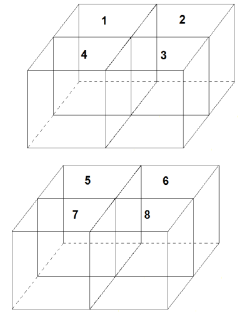

Figure 16. Exemplary eight finite element mesh

sub-page instances of appropriate type. All **P**[#] places in the main page are output sockets, bound to port nodes (places **P63** and **P64**) in sub-pages. Socket nodes in the main page and corresponding port nodes in sub-pages are places by means of which sub-pages are bound to the main page, comprising a coherent model. For precise definitions of socket nodes and port nodes, we refer to [39], page 176. Transitions **Face12**, **Face34**, **Face56** and **Face78** are substituted with instances of a sub-page modeling a mesh element pair that is face-adjacent along X axis. Transitions **Face14**, **Face23**, **Face57** and **Face68** are substituted with instances of a sub-page modeling a mesh element pair that is face-adjacent along Y axis. Transitions **Face15**, **Face26**, **Face38** and **Face47** are substituted with instances of a sub-page modeling a mesh element pair that is face-adjacent along Z axis. Transitions **Edge**[#][#] are substituted with instances of a sub-page modeling a mesh element pair that is edge-adjacent in appropriate direction.

Remark 3. Complexity of the proposed model can be estimated by the number of adjacent element pairs in a computational mesh (directly determining the number of sub-pages in the hierarchical Petri net model). This number is highest for (regular) hexahedral meshes and can be expressed as $((x1)yz+x(y1)z+xy(z1))$ for the number of face-adjacent element pairs, plus $2(((\min(x,y)1)\max(x,y)+\min(x,y)+1)z+((\min(y,z)1)\max(y,z)+\min(y,z)+1)x)$ for the number of edge-adjacent element pairs, where x, y, z denote the number of elements in X, Y and Z directions, respectively.

The number of reachable states of Petri net is computed automatically using the PIPE software. We got the following numbers: 5 391 states for face-adjacent element pair in dead-end prone graph grammar, 3 918 states for edge-adjacent element pair in dead-end prone graph grammar, 5 859 states for face-adjacent element pair in dead-end free graph grammar, and 3 918 states for edge-adjacent element pair in dead-end prone graph grammar.

Remark 4. The grammar is not dead-end-free.

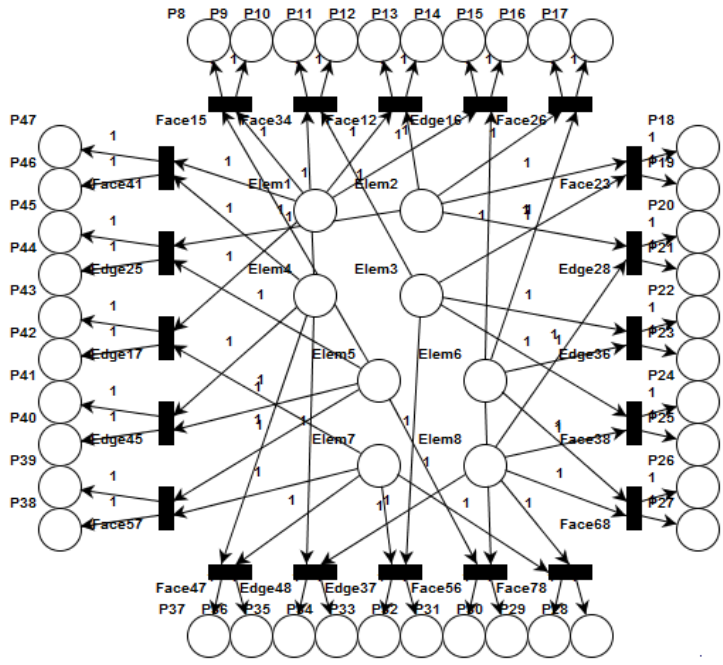

Figure 17. The main page of the hierarchical Petri net for the eight element mesh example

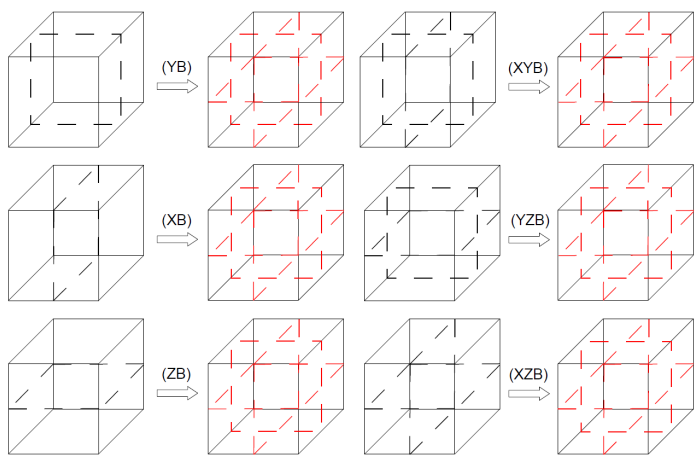

Figure 18. Additional graph grammar productions for removing of the dead-end problem

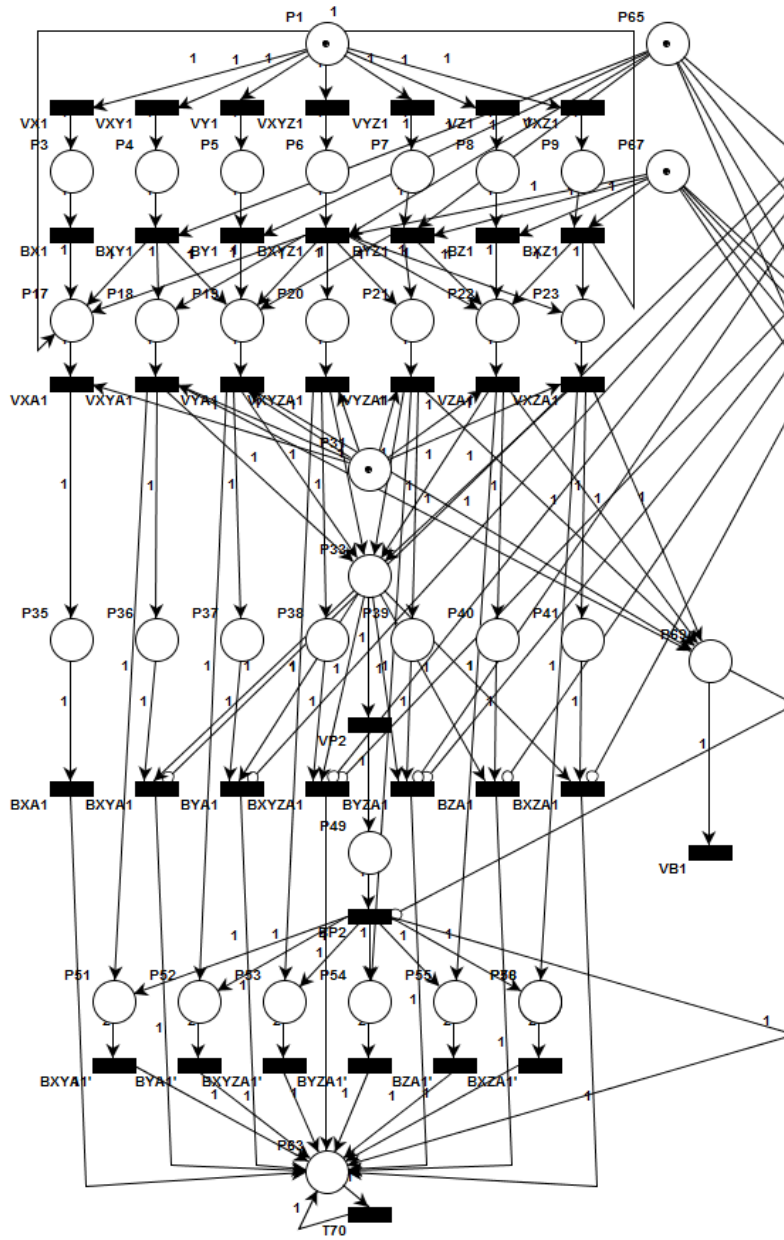

Figure 19. First part of dead-end free hierarchical Petri net for finite elements adjacent along X axis

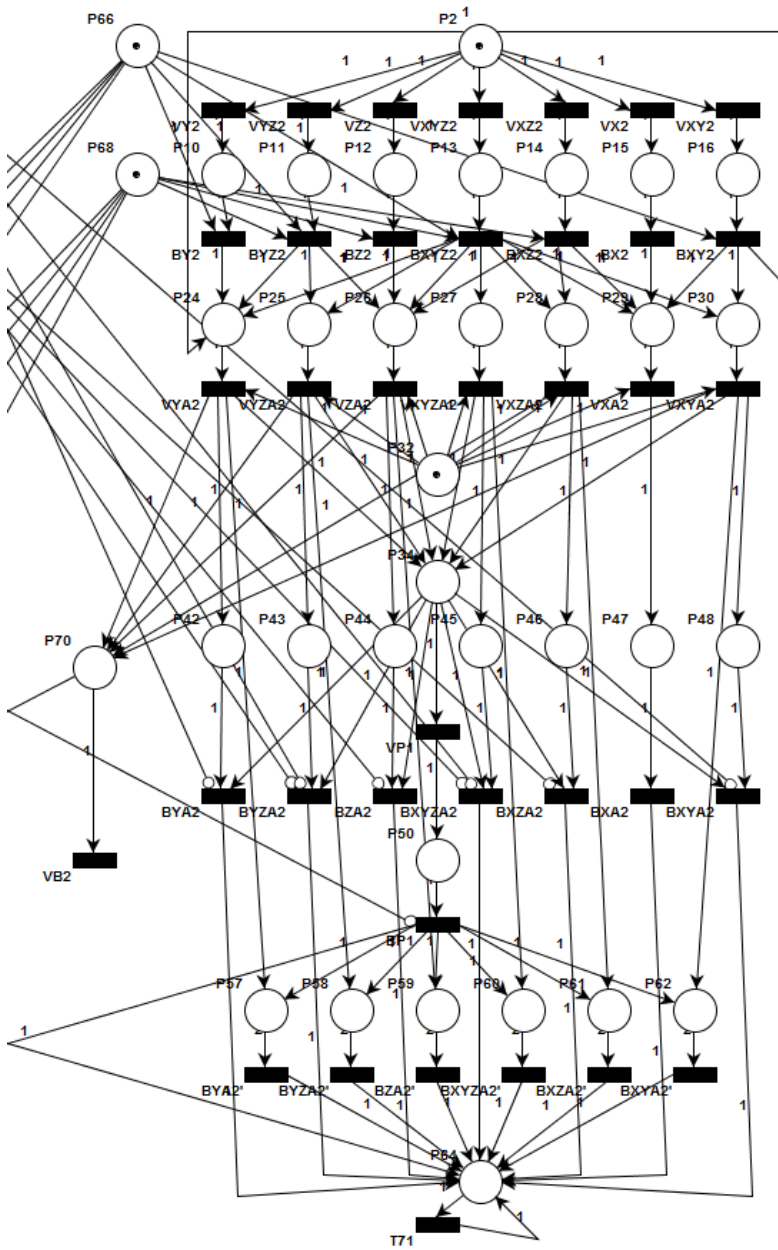

Figure 20. Second part of a dead-end free hierarchical Petri net for finite elements adjacent along X axis

Proof. It is clearly visible that the following sequence of fired transitions (in the subpage for a mesh element pair adjacent along X axis) **VY1**, **BY1**, **VZ2**, **BZ2**, **VYA1**, **VZA2**, **VP2**, **VP1** leads to a dead state. In this state, two mutually contradicting refinement requests have occurred on both pair elements, leading to a dead-end scenario. □

7 ENHANCED GRAMMAR

In this section, we provide some additional graph grammar productions that allows us to overcome the deadlock problem. Figure 18 presents productions that have been added to the previously defined grammar. These graph grammar productions update the broken interior of an element in order to merge two different refinement requests. The implementation of this additional graph grammar productions in the mesh adaptation Algorithm 1 requires to replace the line 18 with the following lines:

```
18a   if  element el interior is already broken then
18b     replace the virtual refinement of element el by the mixture of
18c     actual_refinement_kind and the new refinement kind
18d   else
18e     break element el in a way kind using the virtual refinement
18f   endif
```

Figures 19 and 20 present the counterparts of the dead-end detecting Petri net sub-page reflecting the enhanced grammar, for finite element pairs adjacent along X axis. It is also possible to construct analogous Petri nets for elements adjacent along Y and Z directions, as well as corresponding dead-end free Petri net sub-page for elements adjacent through an edge. Transitions **VB1** and **VB2** have been added to the hierarchical Petri net sub-pages, with arcs from places **P69** and **P70**, respectively. Firing the newly added transitions effectively uninhibits transitions **BP2** and **BP1** respectively, should any of the latter had been previously inhibited (by firing a transition representing a contradicting refinement request). This result demonstrates that executing any of the newly added grammar productions reconciles the contradicting refinement requests. Additionally, arcs **BP2** $\rightarrow$ **P64** and **BP1** $\rightarrow$ **P63** have been added to reflect the fact that actual execution of the propagated refinement brings a given mesh element to the next adaptation level.

Remark 5. The enhanced grammar is dead-end-free.

Proof. Reachability graph has been generated from PIPE [7] for a given Petri net and given initial marking (shown in the figures). The initial marking reflects the intention of breaking each mesh element once. The reachability graph contains no dead state (the Petri net is alive), which implies that the grammar modeled by the Petri net is dead-end-free. □

8 NUMERICAL RESULTS

We have executed the *hp3d* code with the original mesh adaptation algorithm over a problem consisting of the simulation of 3D direct current DC borehole resistivity measurements in deviated wells. A quantity of interest, in this case the voltage, is measured at a receiver electrode located in a logging instrument that is moved along the borehole. Thus, logging instruments are used to estimate the electrical conductivity of the sub-surface material, with the ultimate objective of describing oil or gas bearing formations. In this section, the behavior of a resistivity logging instrument is simulated by performing computer-based simulations of resistivity logging instruments in a borehole environment [23]. The 3D simulations of resistivity measurements in deviated wells, where the angle between the borehole and the formation layers is not equal to 90 degrees, are of particular interest to the oil industry. We consider different electrode configurations (see left panel in Figure 21) and dip angles, which is the angle of incidence between the well and the formation layers.

The strong formulation is the following: Find $u : \Omega \ni x \to u(x) \in R$ where $\Omega \subset R^3$ the electrostatic scalar potential such that $-div(\sigma \nabla u) = \nabla \cdot J$ in Ω, (the conductive media equation), where $\nabla \cdot J$ is the load (divergence of the impressed current) and σ represents the conductivity of the media, defined according to right panel in Figure 21. The electrostatic scalar potential u is related to the electric field E by $E = -\nabla u$. The boundary conditions are defined as $u = 0$ on $\partial\Omega$.

The strong formulation is transformed into the weak one: Find $u \in V$ such that

$$b(u,v) = l(v) \quad \forall v \in V \quad b(u,v) = \int_\Omega \sum_{i=1}^{3} \sigma \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \mathrm{d}x \quad l(v) = \int_\Omega \sum_{i=1}^{3} v \frac{\partial J}{\partial x_i} \mathrm{d}x \quad (5)$$

where $V = H_0^1(\Omega)$.

The adaptive algorithm from the *hp3d* code [9] generates a sequence of computational grids delivering exponential convergence of the numerical error with respect to the mesh size. The sequence of meshes is obtained by performing h refinements (by breaking selected elements in one of eight possible cases) or p refinements (by modifying the polynomial order of approximation on finite element edges, faces, and interiors). This adaptation process is performed by considering a sequence of coarse and fine grids, and by selecting the optimal refinements over the coarse grid resulting from comparison of the coarse and fine grid solutions, compare Figure 22. In this paper we omit the details of the algorithm selecting the optimal refinement, and we refer to [26] for their description.

The original mesh adaptation algorithm from *hp3d* code stopped after executing several h refinements due to a dead-end scenario. This motivated us to construct the graph grammar model of the adaptation algorithm, as summarized in Figure 13, as well as to define Algorithm 2 for constructing hierarchical Petri net based on the nets presented in Figures 14–15, in order to analyze the possibility of a dead-end scenario in the Algorithm 2. The hierarchical Petri net detected a dead-end,

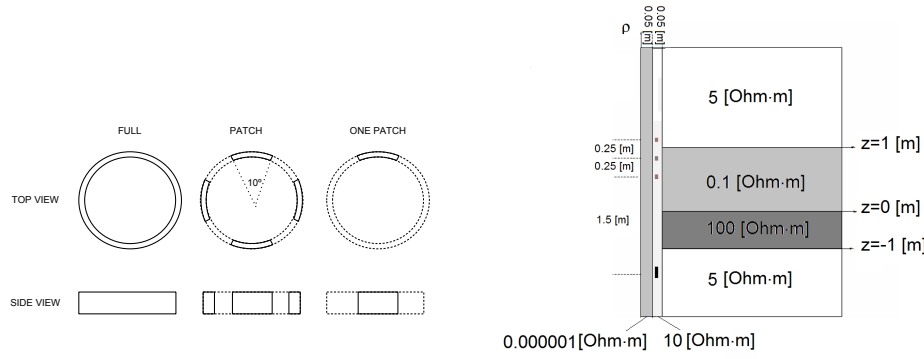

Figure 21. Left panel: The geometry of antennas. Right panel: The conductivities of the borehole, mandrel and formation layers in cylindrical coordinates.

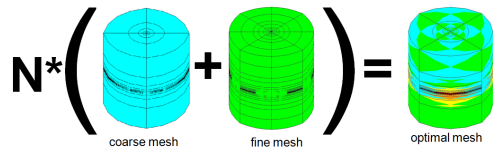

Figure 22. The sequence of coarse, fine and optimal grids generated by the self-adaptive *hp*-FEM algorithm

meaning the adaptation algorithm from [9] needed corrections, and this motivated us to extend the graph grammar with additional graph grammar productions presented in Figure 18. Next, we used the Algorithm 2 for construction of the hierarchical Petri net based on the augmented nets presented in Figures 19–20, including the extended graph grammar model. The new hierarchical Petri net was dead-end free, and this allowed us correct the original adaptation algorithm from *hp3d* and to overcome the dead-end problem as well as to finish the computation process. The problem of propagation of electromagnetic waves has been solved for a sequence of positions corresponding to different locations of the logging tool moving along the borehole, for axial-symmetric as well as 30 and 60 degrees deviated well. The exemplary resulting potential at the receiver electrode for a single position of the tool is presented in Figure 23. The resulting logging curves for different dip angles and different kind of antennas are presented in Figures 24–25.

From the physical point of view, we observe the following. For the axisymmetric case (Figure 33), we obtain a response that is independent of the type of antenna, as physically expected. As we increase the dip angle (Figures 34 and 35), the effect of the antenna becomes noticeable, being the "one-patch antenna" the one that is more sensitive to variations in the formation resistivity.

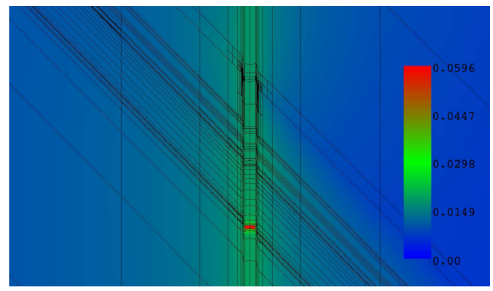

Figure 23. Exemplary resulting potential at the receiver electrode for a single position of a logging tool for 60 degrees deviated well

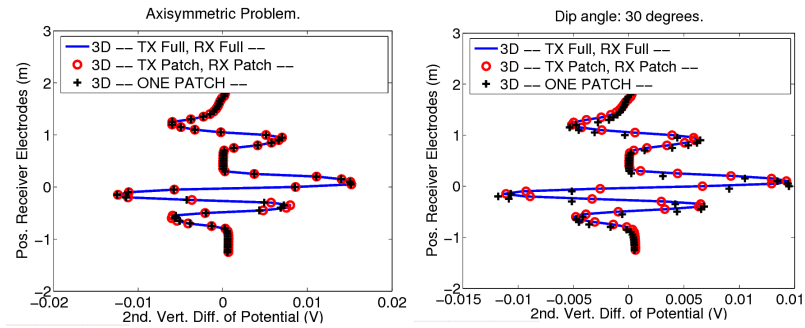

Figure 24. Left panel: Logging curves for different antennas for axial-symmetric case. Right panel: Logging curve for different antennas for 30 degrees deviated well case.

9 CONCLUSIONS AND FUTURE WORK

The main contribution of this paper was the construction of the Petri net model for the analysis of the anisotropic h-refinement algorithm, as implemented in the

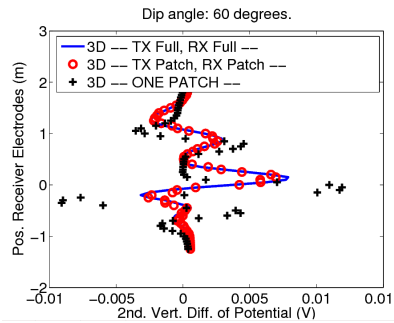

Figure 25. Logging curve for different antennas for 60 degrees deviated well case

self-adaptive 3D FEM package from [9]. The 3D mesh with hexahedral elements was modeled by CP-graph model. The mesh transformation rules were modeled by graph grammar productions. The expression of the mesh adaptation rules by graph grammar productions allowed a construction of the Petri net model expressing the anisotropic mesh refinements algorithm. The Petri net model was implemented in PIPE software, which allowed an automatic analysis of the properties of the adaptation algorithm. In particular, we detected the possibility for a dead-end in the execution of the algorithm. This, in turn, motivated us to construct an improved mesh adaptation algorithm, which was proved to be dead-end free. Based on the corrected model, we fixed the original adaptation algorithm from [9] to make it dead-end free. The practical motivation for this work was the dead-end problem found in the adaptive algorithm during its execution for the borehole resistivity measurement simulation problem. The Petri net model analysis allowed us to correct the adaptation algorithm and, in particular, to finish the adaptive computations with the accuracy required for the borehole resistivity measurements, a problem of great interest to the geophysical community. The future work may involve construction of the Petri net to unstructured 3D grids with hexahedral, tetrahedral, prism and pyramidal elements.

Acknowledgment

The work presented in this paper was supported by Polish National Science Center grant No. NN 519 447739 and DEC-2012/06/M/ST1/00363. David Pardo has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 644602, the Project of the Spanish Ministry of Economy and Competitiveness with reference MTM2013-40824-P, the BCAM Severo Ochoa accreditation of excellence SEV-2013-0323, the CYTED 2011 project 712RT0449, and the Basque Government through the BERC 2014-2017 program and the Consolidated Research Group Grant IT649-13 on Mathematical Modeling, Simulation, and Industrial Applications (M2SI).

REFERENCES

[1] Banaś, K.—Michalik, K.: Design and Development of an Adaptive Mesh Manipulation Module for Detailed FEM Simulation of Flows. Procedia Computer Science, Vol. 1, 2010, No. 1, pp. 2043–2051.

[2] Bao, G.—Hu, G.—Liu, D.: An h-Adaptive Finite Element Solver for the Calculations of the Electronic Structures. Journal of Computational Physics, Vol. 231, 2012, No. 14, pp. 4967–4979.

[3] Babuška, I.—Rheinboldt, W.: Error Estimates for Adaptive Finite Element Computations. SIAM Journal of Numerical Analysis, Vol. 15, 1978, No. 4, pp. 736–754.

[4] Banaś, K.: A Model for Parallel Adaptive Finite Element Software. Proceedings of 15th International Conference on Domain Decomposition Methods, Freie Universität Berlin, 2003.

[5] Becker, R.—Kapp, H.—Rannacher, R.: Adaptive Finite Element Methods for Optimal Control of Partial Differential Equations: Basic Concept. SIAM Journal on Control and Optimisation, Vol. 39, 2000, No. 1, pp. 113–132.

[6] Belytschko, T.—Tabbar, M.: H-Adaptive Finite Element Methods for Dynamic Problems, with Emphasis on Localization. International Journal for Numerical Methods in Engineering, Vol. 36, 1993, No. 24, pp. 4245–4265.

[7] Chung, E.—Kimber, T.—Kirby, B.—Master, T.—Worthington, M.—Knottenbelt, W.: Petri Nets Group Project Final Report, `http://pipe2.sourceforge.net/documents/PIPE2FinalReport.pdf`.

[8] Demkowicz, L.: Computing with hp-Adaptive Finite Elements, Vol. I. Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications. Chapman and Hall/CRC Applied Mathematics and Nonlinear Science, 2006.

[9] Demkowicz, L.—Kurtz, J.—Pardo, D.—Paszyński M.—Rachowicz, W.—Zdunek A.: Computing with hp-Adaptive Finite Elements, Vol. II. Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications. Chapman and Hall/CRC Applied Mathematics and Nonlinear Science, 2007.

[10] Demkowicz, L.—Pardo, D.—Rachowicz, W.: 3D hp-Adaptive Finite Element Package (3Dhp90) Version 2.0. The Ultimate (?) Data Structure for Three-Dimensional Anisotropic *hp*-Refinements. TICAM Report 02-24, 2002.

[11] Demkowicz, L.—Rachowicz, W.—Devloo, P.: A Fully Automatic *hp*-Adaptivity. Journal of Scientific Computing, Vol. 17, 2001, No. 1–3, pp. 127–155.

[12] Devine, K. D.—Flaherty, J. E.: Parallel Adaptive hp-Refinement Techniques for Conservation Laws. Applied Numerical Mathematics, Vol. 20, 1996, pp. 367–386.

[13] Edwards, H. C.—Stewart, J. R.—Zepper, J. D.: Mathematical Abstractions of the SIERRA Computational Mechanics Framework. Proceedings of the Fifth World Congress on Computational Mechanics, Vienna, Austria, 2002.

[14] Errikson, K.—Johnson, C.: Adaptive Finite Element Methods for Parabolic Problems I: A Linear Model Problem. SIAM Journal on Numerical Analysis, Vol. 28, 1991, No. 1, pp. 43–77.

[15] Gomez-Revuelto, I.—Garcia-Castillo, L. E.—Llorente-Romano, S.—Pardo, D.: A Three-Dimensional Self-Adaptive hp Finite Element Method for the Characterization of Waveguide Discontinuities. Computer Methods in Applied Mechanics and Engineering, Vol. 249–252, 2012, pp. 62–74.

[16] Gurgul, P.—Sieniek, M.—Paszyński, M.—Madej, L.—Collier, N.: Two-Dimensional HP-Adaptive Algorithm for Continuous Approximations of Material Data Using Space Projection. Computer Science, Vol. 14, 2013, No. 1, pp. 97–112.

[17] Kardani, M.—Nazem, M.—Abbo, A.—Sheng, D.—Sloan, S.: Refined *h*-Adaptive Finite Element Procedure for Large Deformation Geotechnical Problems. Computational Mechanics, Vol. 49, 2012, No. 1, pp. 21–33.

[18] Kyoungjoo K.: Finite Element Modeling of Electromagnetic Radiation and Induced Heat Transfer in Human Body. Ph.D. Thesis, The University of Texas at Austin, 2013.

[19] Laszloffy, A.—Long, J.—Patra, A. K.: Simple Data Management, Scheduling and Solution Strategies for Managing the Irregularities in Parallel Adaptive hp Finite Element Simulations. Parallel Computing, Vol. 26, 2000, pp. 1765–1788.

[20] Nam, M. J.—Pardo, D.—Torres-Verdin, C.: Simulation of Borehole-Eccentered Triaxial Induction Measurements Using a Fourier hp Finite Element Method. Geophysics, Vol. 78, 2013, No. 2, pp. D41–D52.

[21] Niemi, A.—Babuška, I.—Pitkäranta, J.—Demkowicz, L.: Finite Element Analysis of the Girkmann Problem Using the Modern hp-Version and the Classical h-Version. Engineering with Computers, Vol. 28, 2012, No. 2, pp. 123–134.

[22] Nochetto, R. H.—Siebert, K. G.—Veeser, A.: Multiscale, Nonlinear and Adaptive Approximation. Springer, 2009, pp. 409–542.

[23] Pardo, D.—Demkowicz, L.—Torres-Verdín, C.—Paszyński, M.: A Self-Adaptive Goal-Oriented hp-Finite Element Method with Electromagnetic Applications. Part II: Electrodynamics. Computer Methods in Applied Mechanics and Engineering, Vol. 196, 2007, No. 37, pp. 3585–3597.

[24] Pardo, D.—Torres-Verdin, C.: Sensitivity Analysis for the Appraisal of Hydrofractures in Horizontal Wells with Borehole Resistivity Measurements. Geophysics, Vol. 78, 2013, No. 4, pp. D209–D222.

[25] Pardo, D.—Demkowicz, L.—Torres-Verdín, C.—Paszyński, M.: Two-Dimensional High-Accuracy Simulation of Resistivity Logging-While-Drilling (LWD) Measurements Using a Self-Adaptive Goal-Oriented *hp* Finite Element Method. SIAM Journal on Applied Mathematics, Vol. 66, 2006, No. 6, pp. 2085–2106.

[26] Pardo, D.—Torres-Verdín, C.—Paszyński, M.: Simulations of 3D DC Borehole Resistivity Measurements with a Goal-Oriented *hp* Finite-Element Method. Part II: Through-Casing Resistivity Instruments. Computational Geosciences, Vol. 12, 2008, No. 1, pp. 83–89.

[27] Paszyńska, A.—Grabska, E.—Paszyński, M.: A Graph Grammar Model of the hp Adaptive Three Dimensional Finite Element Method. Part I. Fundamenta Informaticae, Vol. 114, 2012, No. 2, pp. 149–182.

[28] Paszyńska, A.—Grabska, E.—Paszyński, M.: A Graph Grammar Model of the hp Adaptive Three Dimensional Finite Element Method. Part II. Fundamenta Informaticae, Vol. 114, 2012, No. 2, pp. 183–201.

[29] Paszyńska, A.—Paszyński, M.: Application of a Hierarchical Chromosome Based Genetic Algorithm to the Problem of Finding Optimal Initial Meshes for the Self-Adaptive hp-FEM. Computing and Informatics, Vol. 28, 2009, No. 2, pp. 209–223.

[30] Paszyńska, A.—Paszyński, M.—Grabska,E.: Graph Transformations for Modeling hp-Adaptive Finite Element Method with Mixed Triangular and Rectangular Elements. Lecture Notes in Computer Science, Vol. 5545, 2009, pp. 875–884.

[31] Paszyńska, A.—Paszyński, M.—Grabska, E.: Graph Transformations for Modeling hp-Adaptive Finite Element Method with Triangular Elements. Lecture Notes in Computer Science, Vol. 5103, 2008, pp. 604–613.

[32] Paszyńska, A.—Paszyński, M.—Szymczak, A.—Pardo, D.: Petri Nets for Detecting a 3D Deadlock Problem in Hp-Adaptive Finite Element Simulations. Procedia Computer Science, Vol. 9, 2012, pp. 1434–1443.

[33] Paszyński, M.—Demkowicz, L.: Parallel Fully Automatic *hp*-Adaptive 3D Finite Element Package. Engineering with Computers, Vol. 22, 2006, pp. 255–276.

[34] Woźniak, M.—Łoś, M.—Paszyński, M.—Dalcin, L.—Calo, V.: Parallel Fast Isogeometric Solvers for Explicit Dynamics. Accepted to Computing and Informationcs, 2015.

[35] Patra, A. K.: Parallel HP Adaptive Finite Element Analysis for Viscous Incompressible Fluid Problems. Ph.D. Dissertation, University of Texas at Austin, 1999.

[36] Patro, S. K.—Selvam, P. R.—Bosch, H.: Adaptive *h*-Finite Element Modeling of Wind Flow Around Bridges. Engineering Structures, Vol. 48, 2013, pp. 569–577.

[37] Plazek, J.—Banaś, K.—Kitowski, J.: Comparison of Message Passing and Shared Memory Implementations of the GMRES Method on MIMD Computers. Scientific Programming, Vol. 9, 2001, pp. 195–209.

[38] Remacle, J. F.—Li, X.—Shephard, M. S.—Flaherty, J. E.: Anisotropic Adaptive Simulations of Transient Flows Using Discontinuous Galerkin Methods. International Journal for Numerical Methods in Engineering, Vol. 62, 2005, pp. 899–923.

[39] Szpyrka, M.: Petri Nets for Modeling and Analysis of Concurrent Systems. Wydawnictwa Naukowo-Techniczne, Warsaw, Poland, 2008.

[40] Szymczak, A.—Paszyńska, A.—Paszyński, M.—Pardo, D.: Anisotropic 2D Mesh Adaptation in hp-Adaptive Finite Element Method. Procedia Computer Science, Vol. 4, 2011, pp. 1818–1827.

[41] Szymczak, A.—Paszyńska, A.—Paszyński, M.—Pardo, D.: Preventing Deadlock during Anisotropic 2D Mesh Adaptation in hp-Adaptive FEM. Journal of Computational Science, Vol. 4, 2013, No. 3, pp. 170–179.

[42] Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services. `http://www.cs.sandia.gov/zoltan/`.

Arkadiusz Szymczak received his M.Sc. (2001) in mathematics with applications to computer science from the University of Łódź, Poland. Since then he has been working as a software engineer, since 2003 being an employee of Sabre Holdings, currently as a member of Sabre Research Group. He is pursuing his Ph.D. at the Department of Computer Science, AGH University of Science and Technology, Kraków, Poland. His research interests include concurrent and distributed computing.

Maciej Paszyński received his Ph.D. (2003) in mathematics with applications to computer science from the Jagiellonian University, Kraków, Poland and habilitation (2010) in computer science from the AGH University of Science and Technology, Kraków, Poland. In 2003–2005, he worked as a postdoctoral fellow at the Institute for Computational Engineering and Sciences (ICES) at The University of Texas at Austin. In summer 2006 and 2007 he worked as a postdoctoral fellow at the Department of Petroleum and Geosystems Engineering at The University of Texas at Austin. Since 2007 he is a frequent visitor at ICES at The University of Texas at Austin, USA, in Basque Center for Applied Mathematics (BCAM), at the Department of Mathematics, Universidad del Pais Vasco, Bilbao, Spain and at King Abdullah University of Science and Technology (KAUST), Saudi Arabia. He is Associate Professor at the Department of Computer Science, AGH University of Science and Technology of Kraków. His research interests include parallel self-adaptive hp finite element method, parallel direct solvers, models of concurrency and computational science. He has published over 70 publications (Web of Science) and he has given over 100 presentations. Since 2012 he holds a position of vice-director of the Department of Computer Science at AGH.

David Pardo received his B.Sc. degree in mathematics from the University of The Basque Country, Spain, in 2000, and his M.Sc. and Ph.D. degrees in computational and applied mathematics from The University of Texas at Austin, in 2002 and 2004, respectively. Then he worked as a postdoctoral fellow and research associate at the Petroleum and Geosystems Engineering at The University of Texas at Austin during the period 2004–2008. During 2008–2010 he worked as Research Professor and team leader of the group Multiphysics, Inversion and Petroleum at the Basque Center for Applied Mathematics (BCAM). Since September 2009 he is Research Professor at IKERBASQUE, the Basque Foundation for Sciences. Since September 2010 he is also Research Professor at the University of the Basque Country. He has published over 90 publications and he has given over 140 presentations. His research interests include computational electromagnetics, petroleum-engineering applications (borehole simulations), adaptive finite-element and discontinuous Petrov-Galerkin methods, multigrid solvers, and multiphysics and inverse problems.

Anna Paszyńska received her Ph.D. (2007) in computer science from the Institute of Fundamental Technological Research, Polish Academy of Sciences, Warsaw, Poland. She works as Assistant Professor at Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, Krakow, Poland. She was a visiting scientist at King Abdullah University of Science and Technology (KAUST) in Thuwal, Saudi Arabia and The University of the Basque Country and Basque Center for Applied Mathematics, Bilbao, Spain. Her research interests include graph grammar, tree algorithms, and ordering algorithms for multi-frontal solvers. She has published over 20 publications (Web of Science).