ExaQUte

Exascale Quantification of Uncertainties for Technology and Science Simulation

D2.1 Meshing "stub" implementation of the capabilities to be delivered

Document information table

Contract number:	800898
Project acronym:	ExaQUte
Project Coordinator:	CIMNE
Document Responsible Partner:	INRIA
Deliverable Type:	Report, Other
Dissemination Level:	PUblic
Related WP & Task:	WP 2 Task 2.1
Status:	Final Version

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **800898**

Authoring

Prepared by:				
Authors	Partner	Modified Page/Sections	Version	Comments
Algiane Froehly	INRIA			
Vicente Mataix	CIMNE			
Carlos Roig	CIMNE			
Contributors				

Change Log

Versions	Modified Page/Sections	Comments
V1.0	Created document, filling basic information	
V1.1	Added serial mesher API	
V1.2	Added parallel mesher API	
V1.3	Corrected WP and related Task numbers	
V1.4	Added Introduction	

Approval

Aproved by:				
	Name	Partner	Date	OK
Task leader	Algiane Froehly	INRIA	30.7.18	OK
WP leader	Algiane Froehly	INRIA	30.7.18	OK
Coordinator	Riccardo Rossi	CIMNE	30.7.18	OK

Executive summary

This document presents a description of the MMG interface adapted to Kratos as well as its parallel counterpart with ParMMG which will be implemented in task 2.1. In the description are included the following items:

- Description of the requirements of the interfaces;
- Description of the data structures used both in Kratos and MMG;
- Proposal of an initial interface for ParMMG.
- The input needed for the model;
- The definition of the model system matrices;
- The solver and time marching schemes;
- The results obtained, and the link to the 3D detailed model.

Table of contents

List of Figures

List of Tables

Nomenclature / Acronym list

Acronym	Meaning
API	Application Programming Interface
OOP	Object Oriented Programming
BC	Boundary Condition
MPI	Message Pasing Interface
GP	Gauss Point

Table 1: Nomenclature / Acronym list

1 Introduction

This deliverable describes the status of the current implementation of Kratos Mulitphysics [1] and MMG [2] [3] interface to provide an API which will allow to use its meshing capabilities as described in the WP2. Also, we define the first iteration of the future interface for parallel meshing based on ParMMG.

2 Interface Description: existing *MMG* interface

The way the interface will be exposed to *Kratos Multiphysics* is via a remeshing process that will be placed inside *Kratos* own meshing application (`MeshingApplication`) called `MMGProcess`. This process will prepare the input modelpart data and will convert the model into *MMG* format. This conversion will be done using several call to *MMG* exposed methods which are directly wrapped inside *Kratos*.

This *Kratos* process considers as input the modelpart of the problem we are interested in remesh and the configuration parameters that has been read from a `.json` file. This gives us a lot of flexibilty, because the parameters can be changed dinamically without refactoring, so adding functionalities to the process is easier than in a coventional input design.

It is important to notice at this point that *MMG* public **API** makes a distinction between 2D and 3D and offers two sets of different functions which are available to use. From the point of view of *Kratos* such distinction does not exist and the interface will be made in such a way that it remains transparent.

The proposed workflow of the process can be seen in the following workflow where the specific calls to *MMG* are displayed.

⚠ Note that we abstract `MMG2D_` and `MMG3D_` into `MMGXD_` in order to simplify the diagram.

⚠ Note also that `MMGXD_Set_iparameter` and `MMGXD_Set_dparameter` (for integer and doubles) are also abstracted to `MMGXD_Set_Xparameter` for similar reasons.

2.1 `InitMesh`

As its name indciates it will serve as the init function and directly call the init function of *MMG*. This will create and initialize the pointers of the mesh and the solution used bt used by *MMG*. This calls `MMGXD_Init_mesh` from the *MMG* **API** (see Figure 1).

2.2 `InitializeMeshData`

Auxiliar sub model part creation

We will start with the creation of auxiliar sub model parts for flags. These model parts are created in order to preserve the flags of the nodes, elements and conditions after remesh. These auxiliar submodelparts will be removed after remesh.

Colour mapping

After that, we will follow with the creation of a colour map, or model part list. Here we will compute a map that will serve us in order tor preserve *Kratos* sub-modelpart structure. In in our implementations we use a process to set the BC (both Neumann or

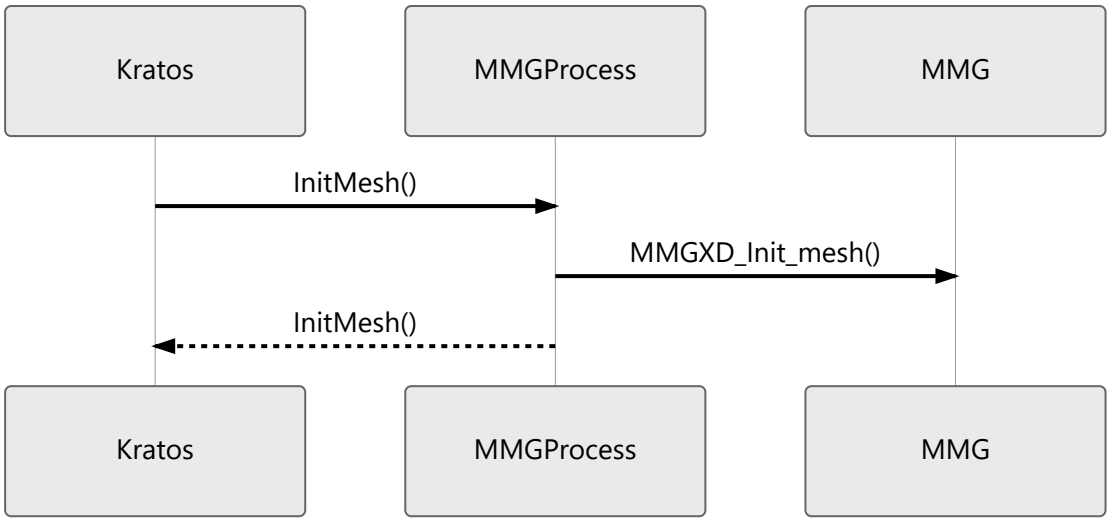

Figure 1: MMGProcess Initialization

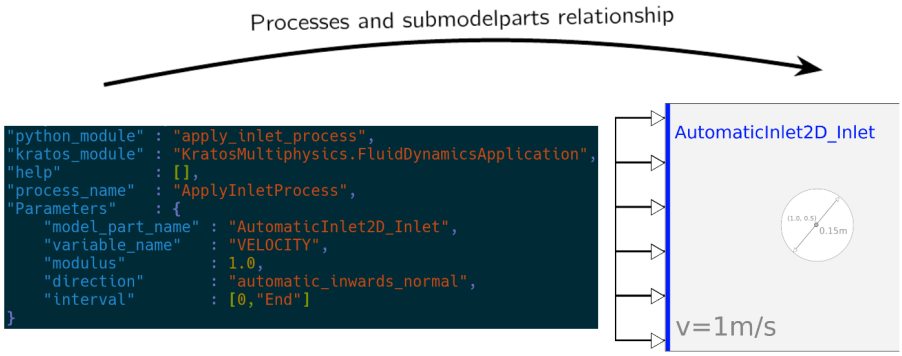

Figure 2: Submodelpart used in order to set **BC**

Dirichlet). We identify the parts where to set this BC with the use of submodelparts (see Figure 2). The same component (node, condition, element) can belong to different submodelparts, so if we want to have an unique ids for each component we need to compute the common belonging to each component. We call this colour see Figure 3.

Node preservation

At this stage, if we want to preseve a part of the mesh we have the option to block individual nodes using the `MMGXD_Set_requiredVertex` function of *MMG*.

Transfer mesh data to *MMG*

In order to be able of doing that we will compute first the number of total nodes, elements and conditions in order to allocate the memory necessary to remesh. Also, we will convert Kratos nodes to *MMG* vertices and finally gemoetries in Kratos to geometries in MMG which will be mapped to edges, triangles, prism and tetrahedras depending on the source geometry (see Figure 4).

- `Kratos_Line2D2` → MMG Edge (`MMG2D_Set_edge`);
- `Kratos_Line3D2` → MMG Edge (`MMG3D_Set_edge`);

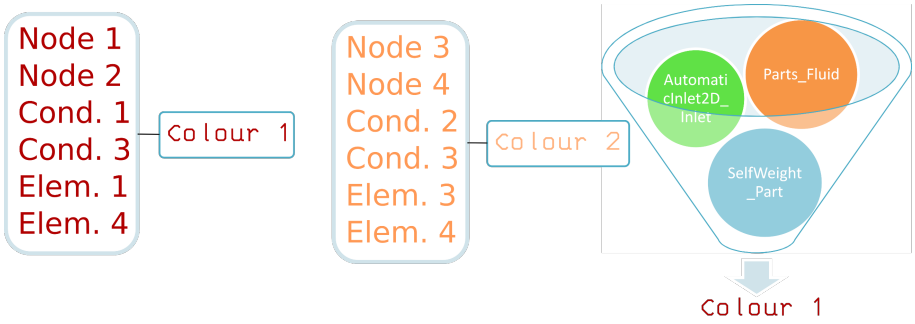

Figure 3: Colour submodelpart creation

- `Kratos_Triangle3D3` → MMG Triangle (`MMG3D_Set_triangle`);
- `Kratos_Quadrialeteral3D4` → MMG Quadrilateral (`MMG3D_Set_quadrilateral`);
- `Kratos_Tetrahedra3D4` → MMG Tetrahedron (`MMG3D_Set_tetrahedron`);
- `Kratos_Prism3D6` → MMG Prism (`MMG3D_Set_prism`);

2.3 `InitializeSolData`

Here we allocate the metric. We always use tensor metrics by default due to its generality allthoug scalar wrapper also exists.

This is done in two differents steps, first we allocate the size using `MMGXD_Set_solSize` and then setting its values using `MMGXD_Set_tensorSol`, `MMG3D_Set_vectorSol` and `MMGXD_Set_solSize` (see Figure 5).

2.4 `CheckMeshData`

Before proceed with the remeshing we call the routines from *MMG* designed in order to check that the information transfered to the **API** of *MMG* is consistent, this is the method `MMGXD_Chk_meshData()`. See Figure 6.

2.5 `ExecuteRemeshing`

We call the remeshing process as well as setting the general configuration values in mmg. At detail, the list of parameters set in this step are the following[1]:

- `MMGXD_DPARAM_hausd` : The *Hausdorff* controls the smothness of the boundary;
- `MMGXD_IPARAM_nomove`: This parameters sets if we allow to avoid or allow the point recolocation;
- `MMGXD_IPARAM_nosurf`: This parameters sets if we don't allow the surfaces modification;
- `MMGXD_IPARAM_noinsert`: This parameters sets if we don't insert nodes on mesh;

[1]These parameters are registered as enums

- `MMGXD_IPARAM_noswap`: This parameters sets if we don't swap the mesh;
- `MMGXD_IPARAM_angle`: This parameters is used in order to deactivate/activate the angle detection;
- `MMGXD_DPARAM_hgrad`: This parameter is used in order to set the gradation;
- `MMGXD_DPARAM_hmin`: This parameters sets the minimum size of the mesh;
- `MMGXD_DPARAM_hmax`: This parameters sets the maximum size of the mesh.

And finally we call the remeshing method `MMGXD_mmgXdlib()` as seen in Figure 7.

After executing the remesh, the new nodes, conditions and elements are created and transfered to the corresponding submodelparts using the respective colours previously computed, which correspond with the *MMG* reference.

Save output mesh and solution

In case we are interested in getting the mesh and solution obtained in the `.mesh` and `.sol` file of *MMG* it exists the possibility to export the new mesh and solution to external files with the use of `SaveSolutionToFile()`. It calls the `MMGXD_Set_outputMeshName` and `MMGXD_Set_outputSolName` methods form the *MMG* **API** to set the filenames and the `MMGXD_saveMesh` and `MMGXD_saveSol` methods to write the output files.

FreeMemory

After we have computed all the relative to *MMG* we can now release the memory relative to it. We use the `MMGXD_Free_all()` method from the `API` for that pourpose. See Figure 8 for more details.

ReorderAllIds

During this step the ids of the element, conditions and nodes are reordered. So the ids will vary from `1` to `n`, being `n` the number of each component. This will not affect to the problem itself, due to the **OOP** nature of *Kratos*, if the ids are reordered, it will be reorder in all the objects depending of it.

This step must be done before and after remesh in order to avoid problematics with the remeshing process. Some steps of the process, including call to `MMG`, could do some assumptions about ordered ids of the different containers of the problem. Then the best way to avoid these problematics will be to reorder the ids, following the previously exposed methodology.

Interpolation of mesh information

This step is necessary in order to preserve the information of the old mesh in the new mesh. We use different processes and utilities in *Kratos* to interpolate both nodeal values and integration values from the old mesh to the new mesh. In order to interpolate internal variables the following procedures are available:

- **Closest Point Transfer**: It transfer the values from the closest GP;
- **Least-Square projection Transfer**: It transfers from the closest GP from the old mesh;

- **Shape Function Transfer**: It transfer GP values to the nodes in the old mesh and then interpolate to the new mesh using the shape functions all the time.

InitializeElementsAndConditions
Finally, the created elements and conditions are initialised in order to be usable in *Kratos*.

3 Interface Proposal: *ParMMG* interface

We propose to update the exiting `MMGProcess` process of Kratos and to couple *Kratos-Multiphysics* and *ParMMG* in the same way as the *KratosMultiphysics-MMG* coupling:

- `MeshingApplication`(*Kratos*) ← `MMGProcess`(*KratosMultiphys.*) ← `MMG API`.
- The parameters and geometry of the problem are given as input for Kratos in a *.json* file.

This section describes the workflow for the *Kratos-ParMMG* coupling.

3.1 `InitMesh`

It is the method used to initialize the `MMGProcess` process. It calls the initialization function of *ParMMG* (`PMMG_Init_mesh`) that creates the pointer toward the main structure of *ParMMG* (a `ParMesh`) and initializes its fields:

- the mesh dimension (unlike *MMG*, *ParMMG* doesn't make distinction between 2D and 3D, thus it is needed to know the mesh dimension in order to call the suitable *MMG* functions for the *MMG* structures initialization);
- a pointer toward a *MMG* mesh;
- a pointer toward a *MMG* metric;
- a pointer toward an array of *MMG* solutions (if needed);
- the MPI communicator in which we will work.

See Figure 9 for the diagramm of the `MMGProcess` initialization.

3.2 `InitializeMeshData`

Auxiliar sub model part creation
This part is similar to the creation of the sub model part with *MMG*.

Node preservation
The `PMMG_Set_requiredVertex` function of *ParMMG* allows to block individual nodes and to preserve a part of the mesh.

Transfer mesh data to *ParMMG*

On each processor, we will need to transfer the local mesh to *ParMMG*.
Again, we will need to compute first the number of nodes, elements and conditions to be able to allocate the memory for the remesher. We set the mesh size calling the `PMMG_Set_meshSize` function. Then, the *Kratos* mesh is converted into a *ParMMG* mesh using the following map:

- `Kratos_Line2D2` or `Kratos_Line3D2` → PMMG Edge (`PMMG_Set_edge`);
- `Kratos_Triangle3D3` → PMMG Triangle (`PMMG_Set_triangle`);
- `Kratos_Quadrialeteral3D4` → PMMG Quadrilateral (`PMMG_Set_quadrilateral`);
- `Kratos_Tetrahedra3D4` → PMMG Tetrahedron (`PMMG_Set_tetrahedron`);
- `Kratos_Prism3D6` → PMMG prism (`PMMG_Set_prism`);

Additionnaly, must provide the communication data to *ParMMG*. Note that *ParMMG* uses the partition of the dual graph of the mesh (partition of the mesh elements) while *Kratos* uses a partition of the mesh (partition of both the mesh elements and nodes). It leads to have more communication informations in *Kratos* than in *ParMMG* (ghost nodes). For this preliminary version of the interface specification, we will only deal with the element partition (the element partition is given to *ParMMG* that gives back the new element partition, leaving to *Kratos* the responsability of the nodes partition).

To set the communication data from *Kratos* to *ParMMG* we need:

- the local to global mapping of the nodes and elements using the `PMMG_Set_loc2GlobVertices` and `PMMG_Set_loc2GlobTetrahedra`;
- for each couple of adjacent processors, the processors IDs and the list of the nodes at their interface (`PMMG_Set_interfaceNodes`).

3.3 `InitializeSolData`

As *ParMMG* modifies both the mesh and its distribution among the processors, it must performs the solutions interpolation over the new mesh. It implies to give the solutions to *ParMMG*.

Thus, the `InitializeSolData` process allocates now the metric and the solutions structure (Figure 11).

Metric allocation

We allocate the metric structure given its size (`PMMG_Set_metSize`), then we set the metric values using `PMMG_Set_tensorMet`.

Solutions array allocation

We allocate the array of solution structure given the number of solutions per entity and the type of each solution (scalar, vectorial or tensorial) using the `PMMG_Set_allSolsSizes` function. Then we set each solution value using the `PMMG_Set_ithSol_inAllSols` function

3.4 ExecuteRemeshing

The `ExecuteRemeshing` process sets the *ParMMG* parameters and run the *ParMMG* library. The list of parameters setted in this step are the following[2]. Lot of them are similar to the *MMG* arguments:

- `PMMG_DPARAM_hausd` : The *Hausdorff* controls the smothness of the boundary;
- `PMMG_IPARAM_nomove`: This parameters sets if we allow to avoid or allow the point recolocation;
- `PMMG_IPARAM_nosurf`: This parameters sets if we don't allow the surfaces modification;
- `PMMG_IPARAM_noinsert`: This parameters sets if we don't insert nodes on mesh;
- `PMMG_IPARAM_noswap`: This parameters sets if we don't swap the mesh;
- `PMMG_IPARAM_angle`: This parameters is used in order to deactivate/activate the angle detection;
- `PMMG_DPARAM_hgrad`: This parameter is used in order to set the gradation;
- `PMMG_DPARAM_hmin`: This parameters sets the minimum size of the mesh;
- `PMMG_DPARAM_hmax`: This parameters sets the maximum size of the mesh.

Because of the mesh repartitionning, the interpolation of the solutions from the old mesh into the new one must be done in *ParMMG*.
The interpolation method is choosen depending on the `PMMG_IPARAM_interp` argument and the wanted method:

- `PMMG_INTERP_linear`: linear interpolation of the solution from the old mesh in the new one;
- `PMMG_INTERP_closestPoint`: transfer of the solution from the closest GP.

Last we call the remeshing method `PMMG_parmmglib_distributed()` as seen in Figure 12.

After the remeshing step, the new nodes, conditions and elements are created and transfered to the corresponding submodelparts using the respective colours previously computed, which correspond with the *PMMG* reference.

Save output mesh, metric and solutions

We can save the mesh, metric and solutions at `.mesh` and `.sol` format (*PMMG* format) using the `SaveSolutionToFile()` process. It calls the `PMMG_Set_outputMeshName`, `PMMG_Set_outputMetName` and `PMMG_Set_outputSolsName` methods form the *PMMG* **API** to set the filenames and the `PMMG_saveMesh`, `PMMG_saveMet` and `PMMG_saveAllSols` methods to write the output files.

FreeMemory
We release the *ParMMG* memory calling the `PMMG_Free_all()` method. See Figure 13 for more details.

ReorderAllIds The only reordering needed from *Kratos* is before remeshing in order to avoid potential problematics. After remesh *Kratos* will take the ids provided by *ParMMG* which already computes an internal reordering.

InitializeElementsAndConditions
Finally, the created elements and conditions are initialised in order to be usable in *Kratos*.

[2]These parameters are registered as enums

References

[1] P. Dadvand, J. Mora, C. González, A. Arráez, P. Ubach, and E. Oñate. Kratos: an object-oriented environment for development of multi-physics analysis software. In *World Congress on Computational Mechanics*, pages 485–485, Jul 2002.

[2] C. Dapogny, C. Dobrzynski, and P. Frey. Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. Technical report, Mar. 2013. URL `https://hal.sorbonne-universite.fr/hal-00804636`.

[3] C. Dobrzynski and P. Frey. Anisotropic delaunay mesh adaptation for unsteady simulations. In *Proceedings of the 17th international Meshing Roundtable*, pages 177–194. Springer, 2008.

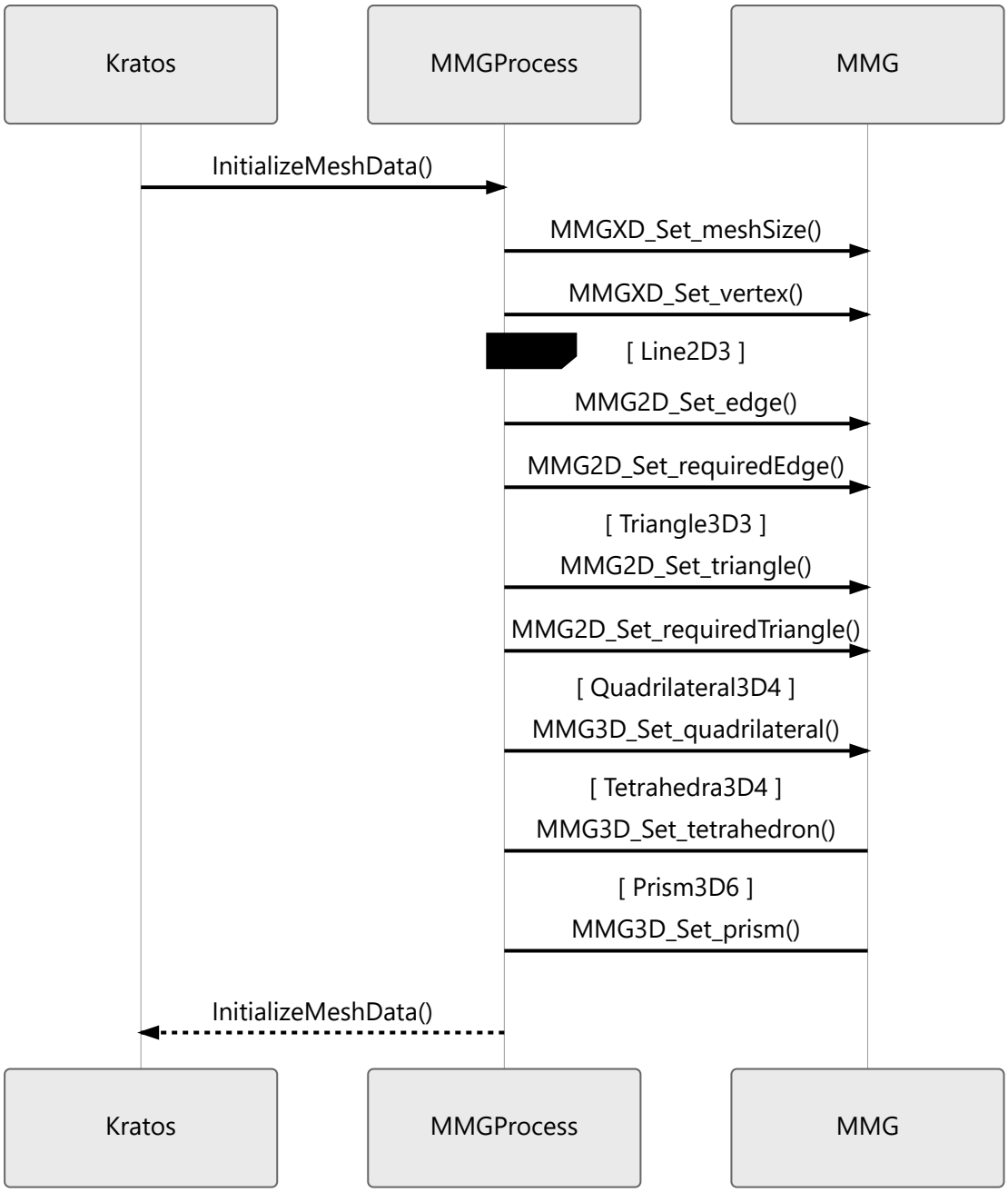

Figure 4: MMGProcess Initialize Mesh Data

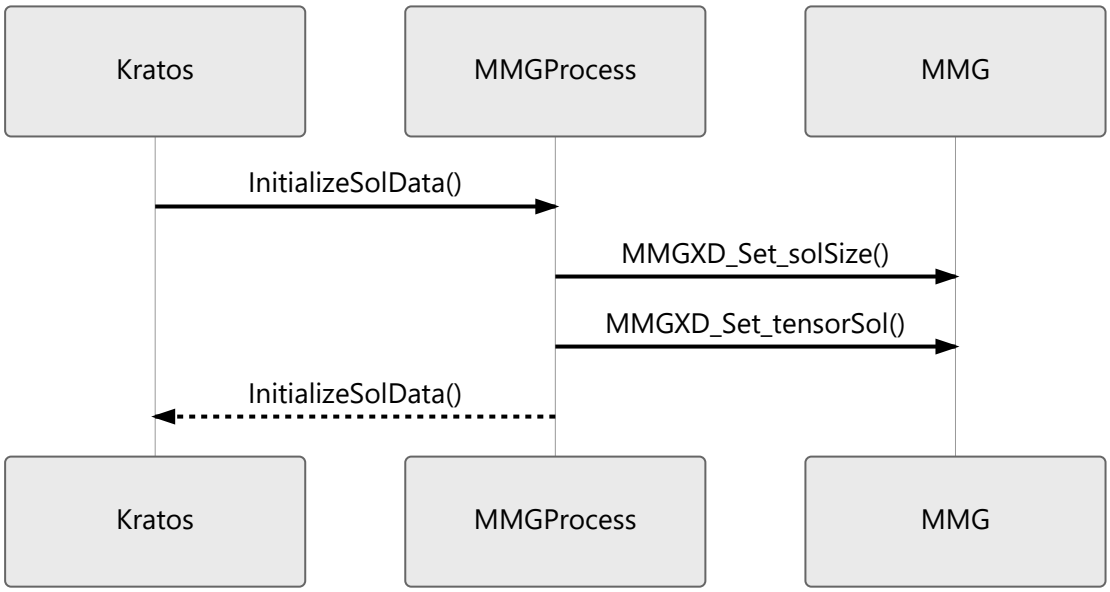

Figure 5: MMGProcess Initialize Solution Data

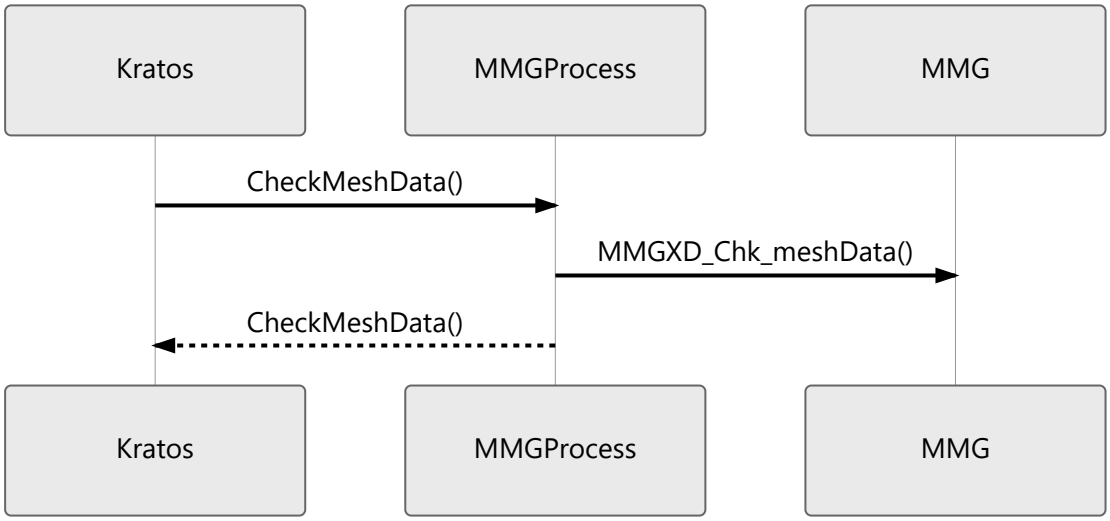

Figure 6: MMGProcess Check

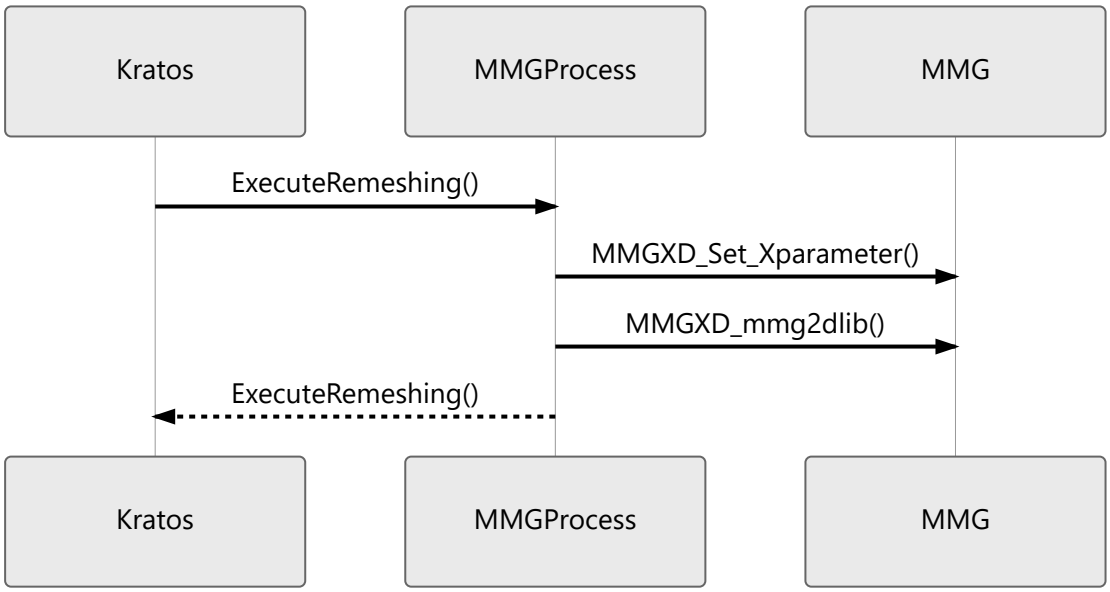

Figure 7: MMGProcess Remesh

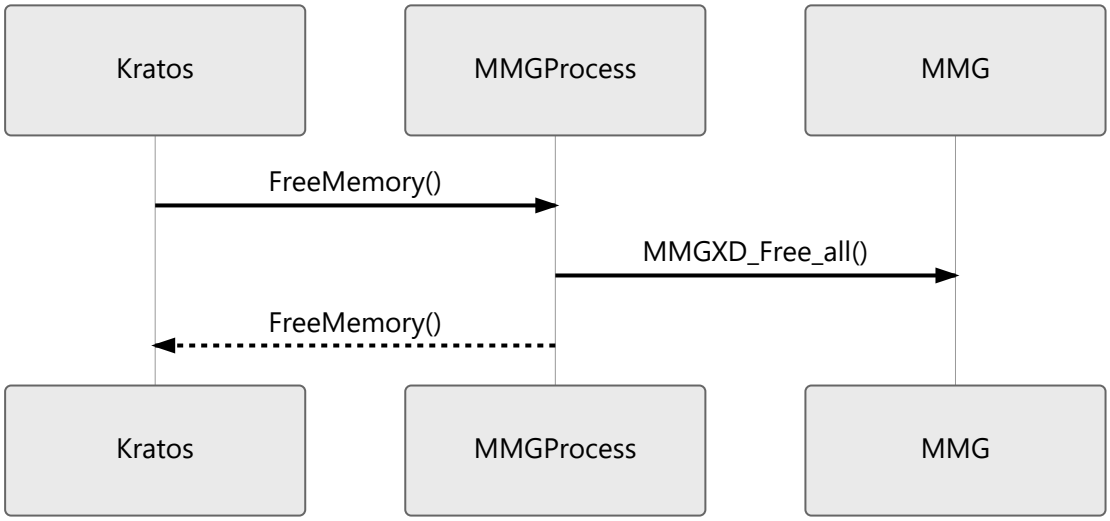

Figure 8: MMGProcess Finalize

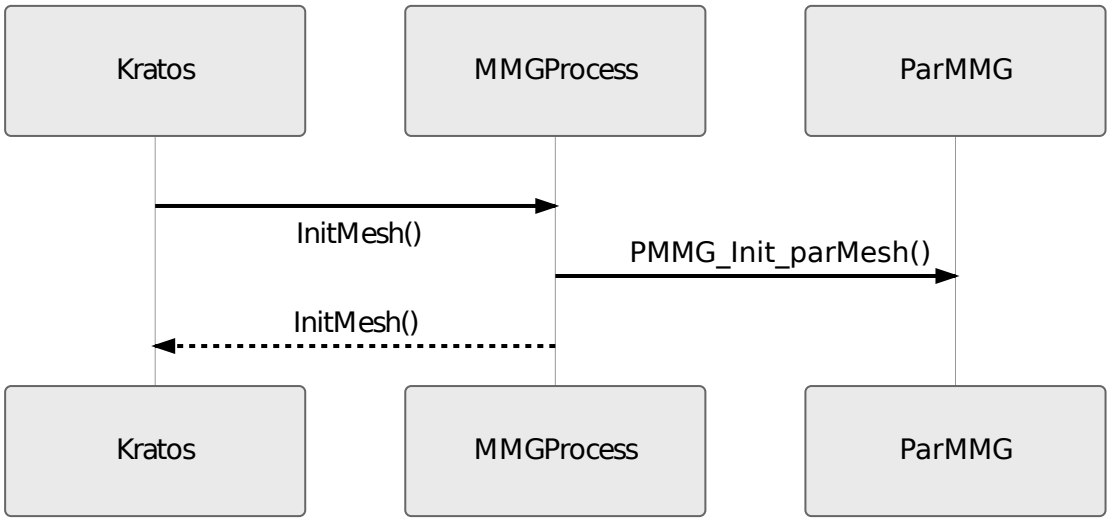

Figure 9: ParMMGProcess Initialization

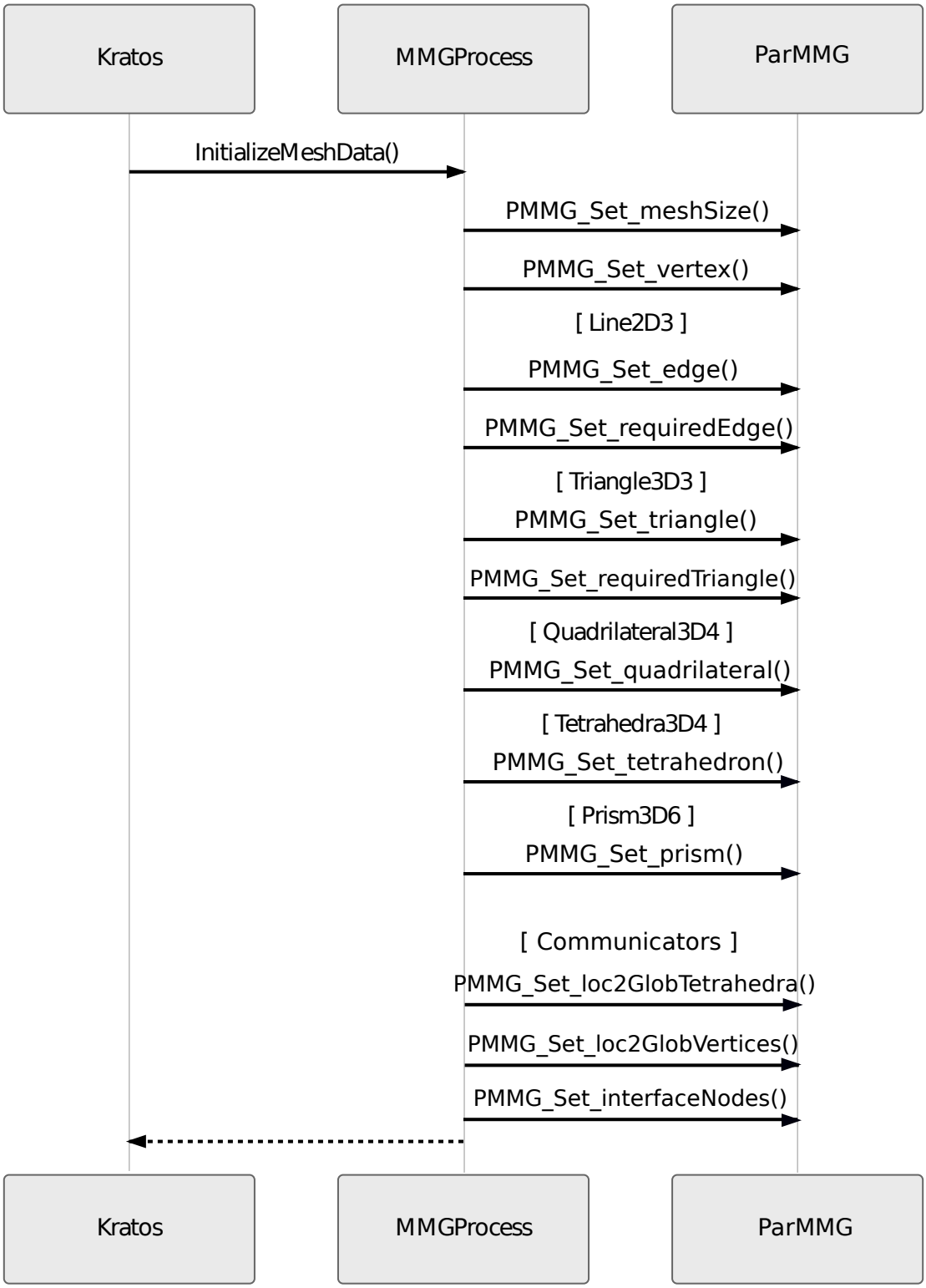

Figure 10: ParMMGProcess Initialize Mesh Data

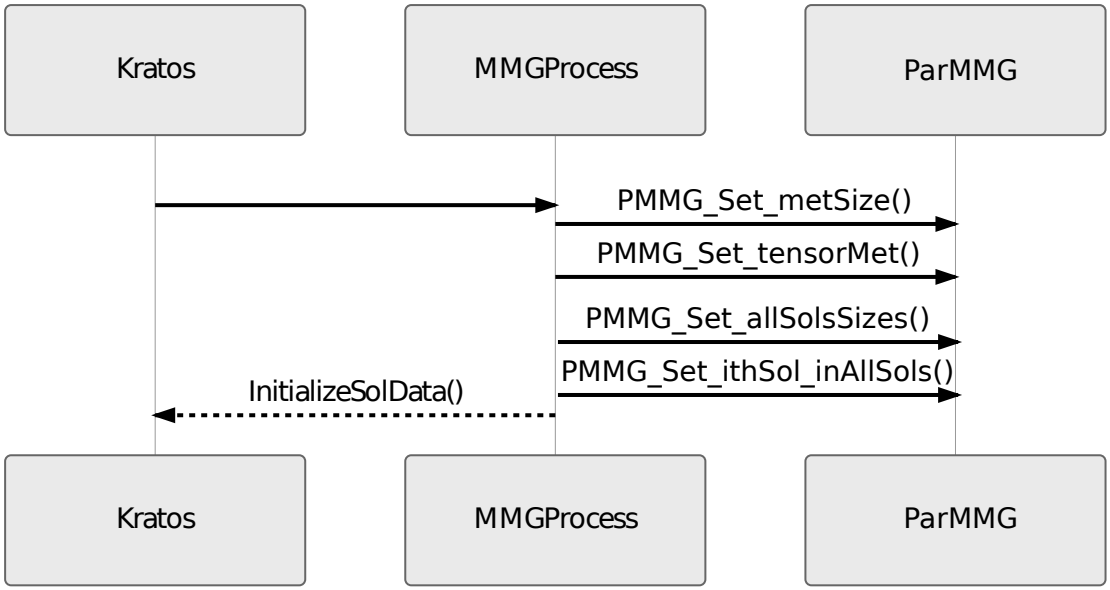

Figure 11: ParMMGProcess Initialize Metric Data

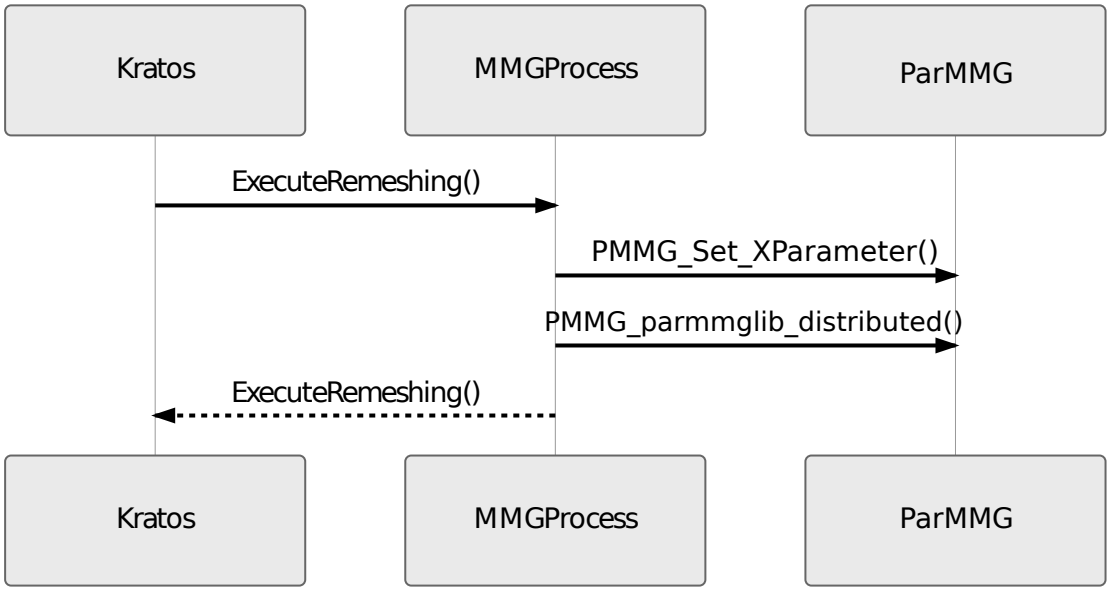

Figure 12: ParMMGProcess Remesh

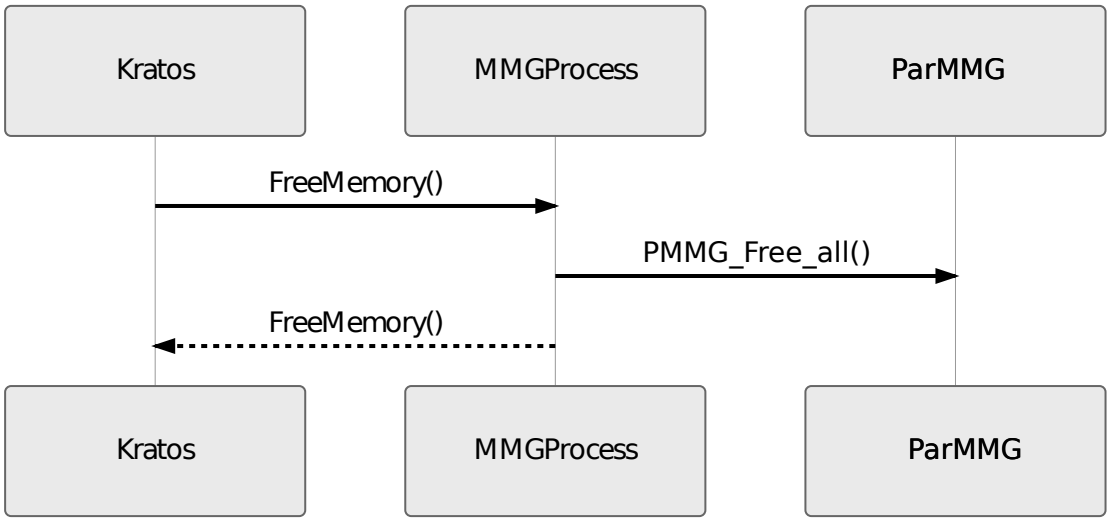

Figure 13: ParMMGProcess Finalize