¡¡DELIVERABLE UNDER REVIEW PROCESS!!

VELaSSCo

***V**isual Analysis for **E**xtremely **La**rge-**S**cale*
***S**cientific **Co**mputing*

D1.3 – Technical requirements

Description and requirements of components and tools, their interaction and communication between them and the final user for the prototype

Deliverable Information

Grant Agreement no	619439
Web Site	http://www.velassco.eu/
Related WP & Task:	WP1 - Steering on target users and general VELaSSCo's requirements
Due date	31/03/2014
Dissemination Level	Public
Nature	Report
Author/s	Miguel Pasenau de Riera, Stefanos Papanicolopulos, Abel Coll, Andreas Dietrich, Frank Michel, Alvaro Janda, Jin Y. Ooi
Contributors	

Approvals

	Name	Institution	Date	OK
Author		**CIMNE**		
Task Leader		**CIMNE**		
WP Leader		**UEDIN**		
Coordinator		**CIMNE**		

Table of Contents

1 Introduction

A simulation of a physical process is performed by solving the equations describing this process using a discretization of the domain of the problem, depending on the method used to solve these equations. This document describes the preliminary technical requirements for achieving the VELaSSCo platforms for visual-analysis of extremely large and complex engineering simulation data. These technical requirements are based on the end-user requirements described in deliverable D1.1 and they are defined to fulfil the functional requirements from an external point of view. They include descriptions of the components and tools, the interaction and communication between them and the final user for the VELaSSCo prototype.

1.1 FEM simulations

In a Computer Fluid Dynamics (CFD) simulation, the domain, for instance, is the air surrounding an F1 racing car. To get this domain, the car is geometrically modelled and placed inside a box. The volume of the box, minus the volume of the car, is the air domain used in the computation. This complex domain is subdivided into very simple geometric elements, usually tetrahedra, sharing their vertices, called nodes, filling the space between the car's surface and the walls of the box. This discretization is called mesh.

After setting the boundary conditions of the simulation (such as initial pressure, velocity, direction of the air, impenetrability of some surfaces) and the properties of the materials (like air density or viscosity), and mathematical model to be used, the calculation process is launched. The simulation program calculates the unknowns of the equations (mainly velocity and pressure), on all the nodes of the mesh.

As the air flow around a car is a dynamic process, the simulation is performed along a time interval and at certain time-steps the results are written in files.

Small CFD simulations can be successfully be calculated with meshes with a few hundred thousand elements to a few million elements on a single computer.

More accurate and complex CFD simulations require from tens of millions to a billion elements.

To solve such big problems, the simulation cannot be run on a single computer. To be able to run the simulation on a pool of computers, a HPC cluster, the domain is partitioned into several sub-domains, usually one for each node of the cluster. That means that the initial mesh of a billion elements is sub-divided into smaller sets and distributed across the cluster. The simulation program, which runs on each node of the cluster, reads their partition and communicates with its siblings to solve the problem. Then, at given time intervals, the program writes its mesh portion and results into its own file.

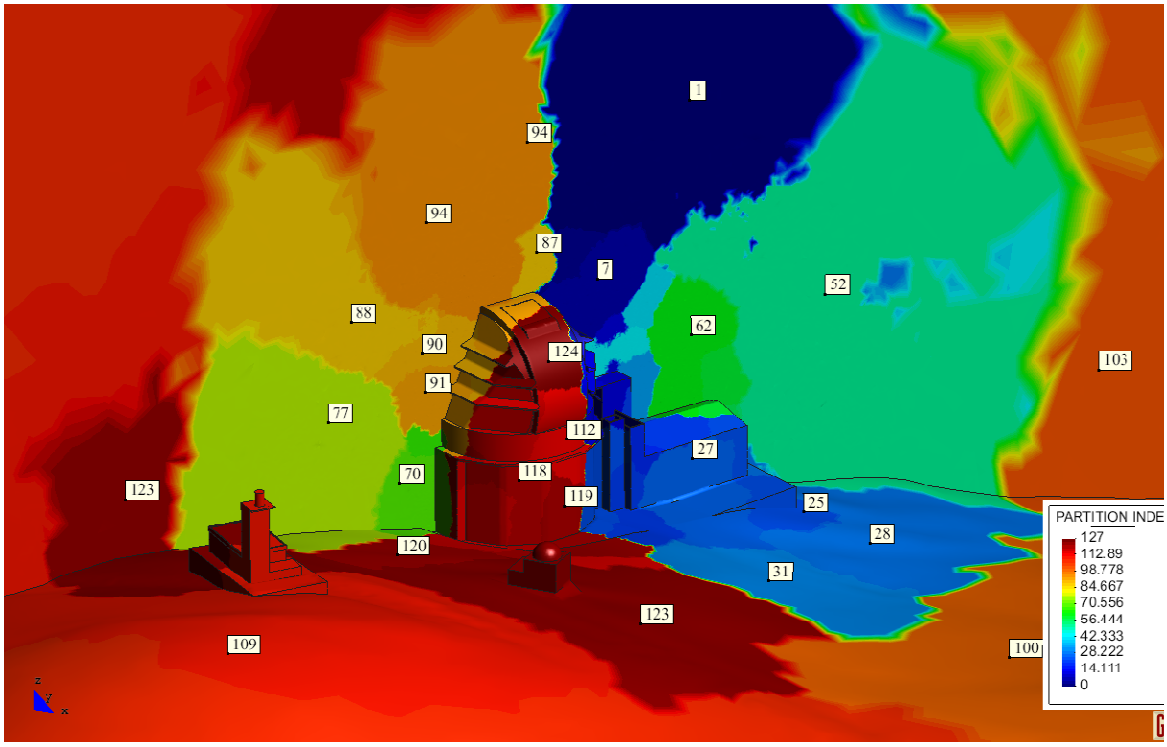

Figure 1.1 example of domain decomposition, the mesh of the air surrounding the telescope, with 24 million tetrahedrons is partitioned into 128 sub-domains, which are coloured in the picture above. The partitions are localized on different nodes of the cluster.

In Figure 1.1 nodes are drawn according to the partition they belong. From the original mesh, elements are distributed into partitions. Two neighbouring elements may belong to two different partitions and the nodes they share are assigned to both partitions, and, thus, duplicated.

1.2 FEM queries

One of the most performed queries by FEM users is to load the simulation data, and visualize the results of the last time-step as contour fill or vectors over the skin of the volume model. Sometimes the vectors displayed belong to the nodes of the volume mesh but with a filter factor, i.e. instead of drawing all vectors which will produce a cluttered view, for instance, one out of 10 are drawn.

These queries only access the results values to draw them as colours or vectors.

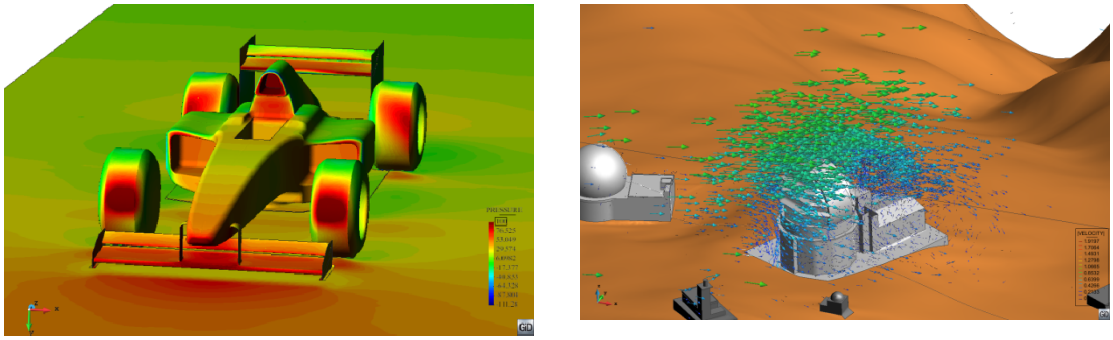

Figure 1.2 simple visualization examples: contour fill of pressure on the left and velocity vectors distribution on the right, one out of hundred vectors are drawn.

Then more complex queries are performed such as defining cut planes to interpolate results from the volume mesh into the plane, or stream-lines, to get a feeling of the vector field of a result into the volume mesh. Also iso-surfaces are used by the users, for instance, to get the free surface of a flow (the iso-surface of pressure equal to 0 extracted from the volume mesh) or the boundaries of embedded objects.

These queries operate at mesh level, by creating a new mesh of lower order than the original (surface or line mesh) from the original one using geometrical or result definition (cut plane coefficients, line definition, result value filter or another result criteria).

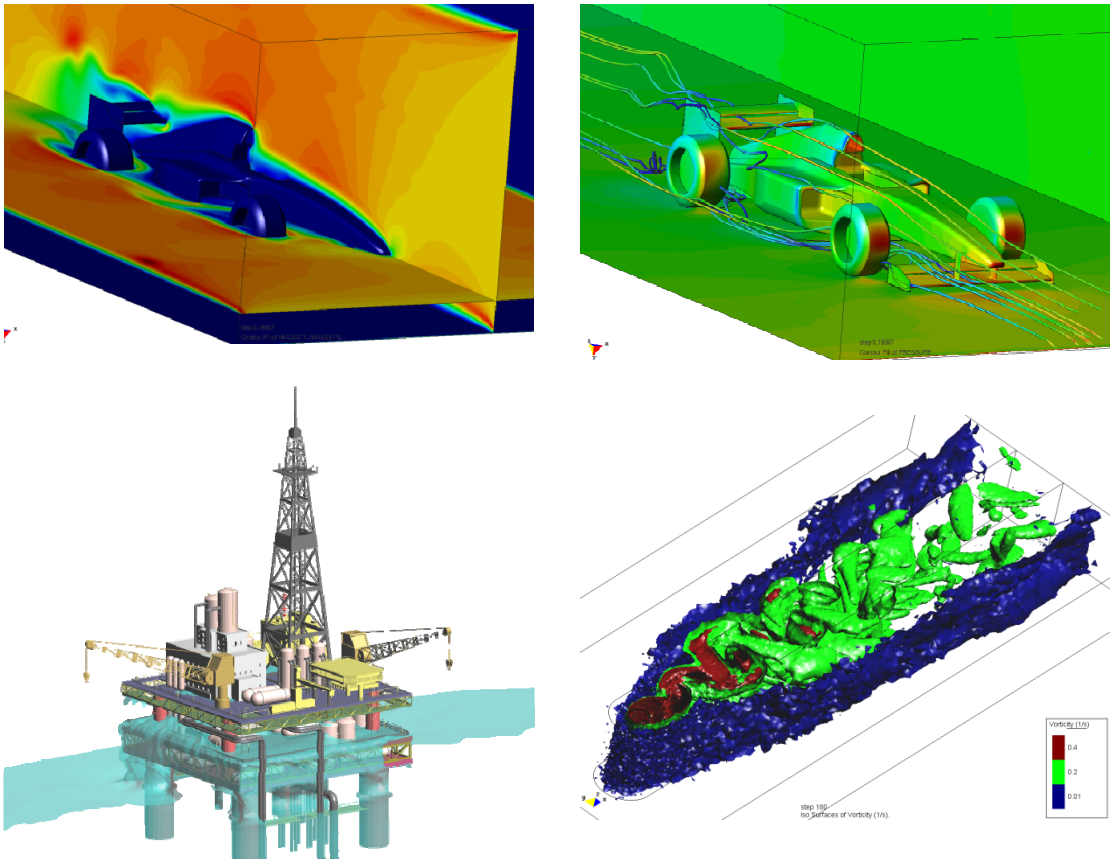

Figure 1.3 simple visualization examples

Also animations of the above visualizations are performed, thus repeating the queries at some or all time-steps.

A more detailed list of the required queries can be found section on FEM – Continuum data User requirements (FEM Analysis) in on the D1.1 deliverable.

1.3 DEM simulations

The Discrete Element Method models the interaction between individual particles through the contacts with their neighbours. DEM primary results are therefore "discrete" results relating either to individual particles or to individual contacts:

- Particle results relate an individual particle ID (PID, a positive integer) to scalar and vector quantities such as mass, position or velocity.
- Contact results relate a contact, identified by the PID of two particles, to scalar and vector quantities such as position or contact force.

DEM computations are always dynamic, computing a large number of successive time-steps, although results are saved at a lower frequency.

Contacts are created and lost throughout the analysis, with consecutive time-steps (or at least consecutive saved time-steps) possibly differing significantly in the particles being in contact.

On the other hand, particles are more persistent within an analysis. Even though particles can enter and exit the analysis, a particle is expected to be present throughout many different time-steps. Some particle quantities such as mass and radius (for spherical particles) are usually constant within the analysis, while other particle quantities, such as position and velocity change in each time-step. Therefore, while individual particle results should contain information about the time to which they refer, as well as to the mass and size of the particle, it can be inefficient to do so. For example, it is usual to group the results by time (so that all results in a single time-step are grouped together).

Most DEM codes only consider particles which are spherical and therefore their shape is identified uniquely by their radius. To model non-spherical particles, it is common to lump together a number of spherical particles using rigid links between them to create a composite particle. In this case, results are still computed at the spherical-particle level, but should be saved and visualised at the composite-particle level. In some cases it is possible to model grain breakage by allowing the rigid links to be broken, so that composite particles can disappear (with new, smaller ones possibly appearing) even without a change in the underlying spherical particles.

1.4 DEM queries on discrete data

The basic query on discrete data regards the position of the particles, with the visualisation showing the position and size of each particle. This is often animated over time to show the displacement (and velocity) of the particles. Colour can also be used to denote scalar particle quantities (e.g. velocity magnitude, or total force on the particle), or vectors can be added to describe vector fields.

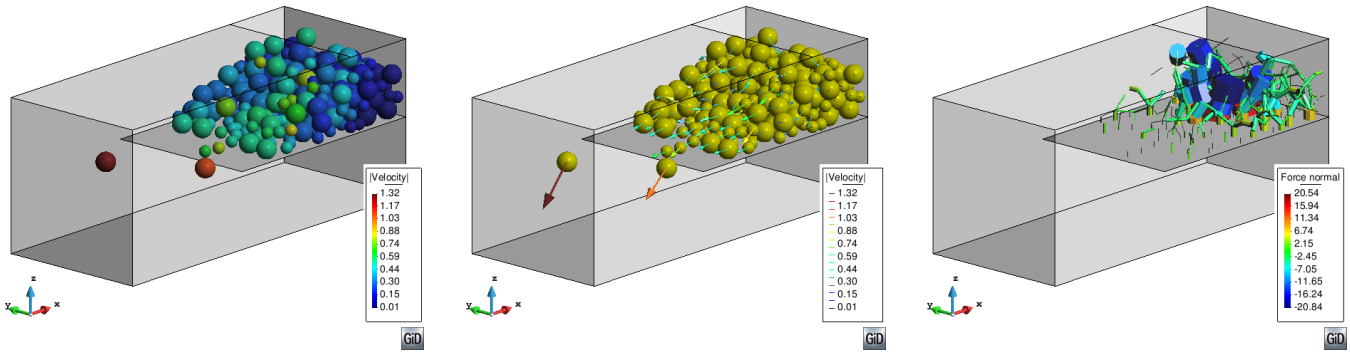

Figure 1.4 simple visualization examples: colour-coded particles on the left, particles with velocity vectors in the centre and normal forces network on the right

Contact discrete data can also be directly queried and visualised, e.g. in the form of contact force networks.

Queries can also regard statistical distributions of quantities for individual particles along several time-steps or across all particles (or contacts) in a given time-step (or time-averaged across more time-steps).

In many practical cases, however, it is not of actual interest to track individual particles or contacts, but to track the overall material behaviour within a domain. A different type of query is then needed on discrete data that computes a “spatial average or coarse graining” and thus projects the discrete data on a continuum field. Through these queries, therefore, we obtain secondary results of a continuum type (similar to FEM and CFD results).

1.5 Continuum results and queries on DEM

The continuum results obtained by coarse-graining can be of the same type as the respective discrete results (e.g. velocity) or they can be of new types (e.g. density, stress, strain – the last two being rank-two tensors). Like discrete data, continuum data can represent a single time-step or a time average of different time-steps.

As in continuum simulations (FEM, CFD) the continuum results for DEM are actually “discretized”, i.e. computed on individual points of a mesh. Indeed, it is useful to run

new queries on the continuum data of the same type as those found in FEM post-processing (e.g. point temporal evolution, line plots, iso-surfaces or slices) including pseudo-colour visualisation.

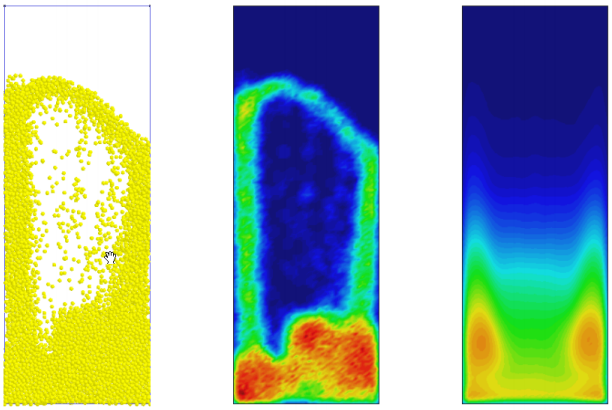

Figure 1.5 simple visualization from the discrete data showing particles on the left, to the continuum results on a static mesh in the centre, showing the spatial average, and on the right image, showing the spatial and temporal averages

A more detailed list of the required queries can be found in sections DEM-discrete data User requirements (Visualisation and DEM analysis) on the D1.1 deliverable.

1.6 Visualization pipeline

In Figure 1.6 a generic visualization pipeline is represented, in order to better understand the data and processes involved in the project.

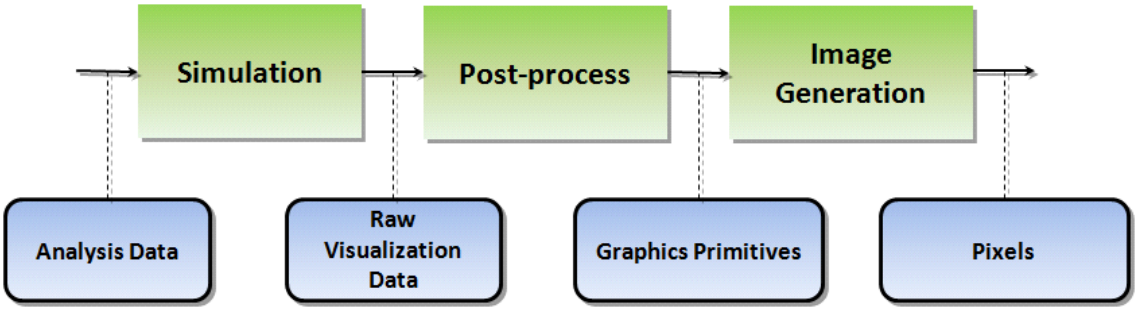

Figure 1.6 generic visualization pipeline, from the data that describes the problem to be simulated to the final images shown in the visualization client

The pipeline shows the usual workflow of the simulation engineer. First the problem is modelled, discretised and their parameters are set: the *Analysis Data*. The simulation

program executes the modelled physical process virtually during a defined time-period, and at certain time-steps outputs the raw results: *raw visualization data*.

When the simulation finished, the simulation engineer downloads the simulation results into their computer and, using a post-processing program, like GiD, RPE/iFX, ParaView, VTK, processes, analyzes and extracts useful information, artefacts like iso-surfaces (surfaces that connects points of space with the same result value), stream-lines, graphs, etc., from the raw simulation data. These artefacts are usually represented by *graphics primitives*, like lines, and triangles, and rendered to an image shown to the user.

The scope of the projects does not include the execution of the simulation on an HPC, but the visualization of the initial data for the simulation, running post-process algorithms, which transforms the raw data from the simulation program into visual artefacts, and the rendering of these artefacts (collection of graphics primitives) on the user's computer.

Due to the size of the raw visualization data, the post-process algorithms cannot be run on a single workstation, and must be run on a distributed environment. To guarantee interactive rates, the post-process can be done in the user's visualization computer on a reduced model or over a region of interest so that the user is able to interact with the data and to control the parameters of the post-process algorithms.

2 Dataset description

Datasets in simulation consists in two distinct parts:

- mesh: which defines the domain and
- results: attributes defined over the mesh.

From a very simplified point of view there are two types of simulations: one in which the mesh is fixed along the whole analysis and the results are defined for several time-steps, and another one in which at each time-step of the analysis, both mesh and results are defined.

The results of the provided FEM examples are defined on the nodes, despite this, results on elements (at the gauss integration points defined on them) should be supported by the VELaSSCo platform too.

2.1 FEM simulation data

Data types in FEM simulation data are:

- **nodes:** defined by an id with their coordinates,
- **elements**: defined by an id and their connectivity, i.e. list of nodes' id that defines the element,
- **meshes**: groups of elements, of the same type, which can also be seen as layers,
- **results**: (in our case defined on nodes) with nodes id followed by their values. Results types are: scalar (1 component), vector (3 or 4 components) and matrices (6 components).

In the FEM cases selected to test the VELaSSCo platform, the mesh is fixed and defined at the beginning of the analysis and at each time-step only the results are written.

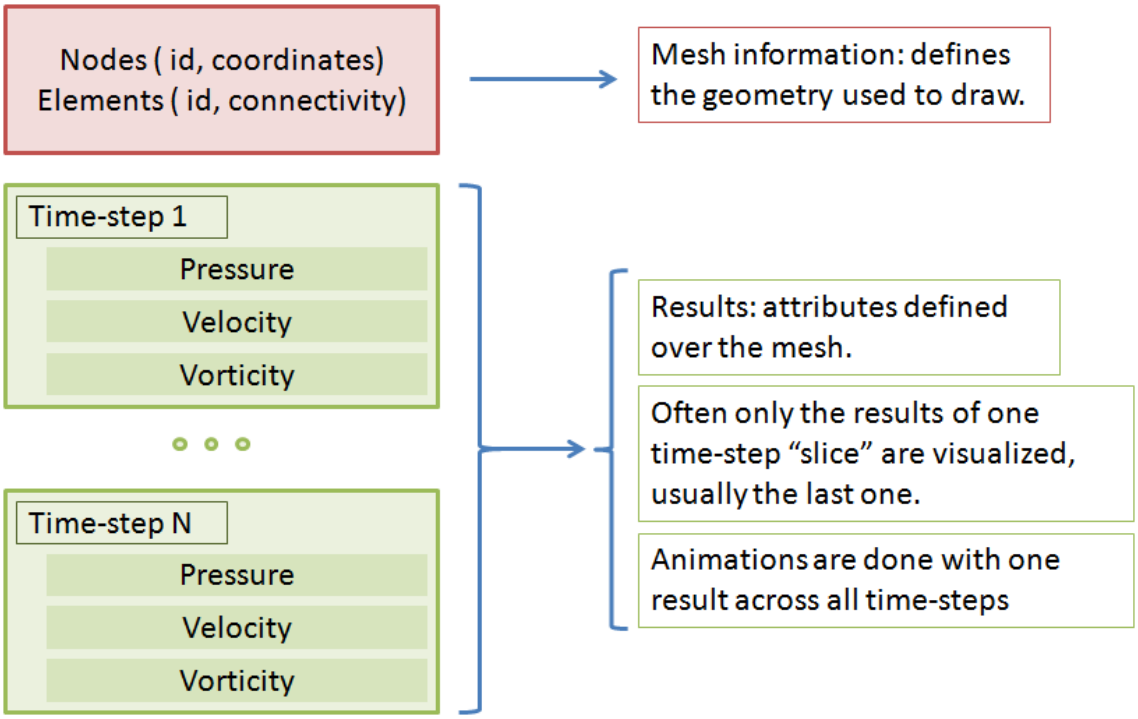

Figure 2.1 Simulation data structure of a FEM simulation run on a single machine.

Mesh description consists of a header followed by the definition of the nodes and their coordinates and the definition of the elements with their node connectivity:

```
MESH "Walls" dimension 3 ElemType Triangle Nnode 3
# color 127 127 10
Coordinates
1 0 100 0
2 10 100 0
…
1116 94.29 75.234 0
end coordinates

Elements
# Element_Id Node_1 Node_2 Node_3
1 30 28 27
2 30 29 28
…
192 65 35 34
end elements

MESH "Another surface" dimension 3 ElemType Quadrilateral Nnode 4
# color 127 200 127
Coordinates
# defined in previous MESH definition
end coordinates

Elements
# Element_Id Node_1 Node_2 Node_3 Node_4
193 100 90 73 23
194 1 4 5 9
```

```
…
232 65 35 34 13
end elements

MESH "Particles" dimension 3 ElemType Sphere Nnode 1
# color 63 180 210
Coordinates
# defined in previous MESH definition
end coordinates

Elements
# Element_Id Node_Id Radius
1000 10 1.12
…
1116 1045 2.33
end elements
```

Listing 2.1 Example of mesh definition using GiD ascii format. Three sets are defined: 'Walls', 'Another surface' and spherical 'Particles'

Like the mesh, the result description consists of a header, followed by a list of pairs of node number and result values.

```
GiD Post Results File 1.0

Result "DESPLACEMENTS" "LOAD ANALYSIS" 10 Vector OnNodes
ComponentNames "X-DESPL", "Y-DESPL", "Z-DESPL", "|DESPLACEMENTS|"
ResultRangesTable "My table"
Values
1 0 0 0 0
2 2.0855e-05 -1.9174e-05 0 2.83297e-05
…

1116 2.4357e-05 -0.00018974 0 0.000191297
End values

Result "Damage factor" "LOAD ANALYSIS" 10 Scalar OnNodes
Values
1 0
2 2.0855e-05
…
1116 2.4357e-05
End values
```

Listing 2.2 Example of result definition using GiD ascii format. Two results are defined: a 'DESPLACEMENTS' vector and 'Damage factor' scalar.

2.2 DEM simulation data

In DEM simulations, every time-step a new particle and contact point mesh is calculated and written with their results. So the result “time-step slice” consists both of mesh (particles and contacts) and results.

After the simulation is run, in addition to visualising particle and contact data, a post-processing algorithm is performed in which the particle information, position and results are statistically analyzed and interpolated to a static mesh. This coarse-graining post-processing should also be included in the VELaSSCo platform. An application that uses this kind of algorithms is the P4 tool. And the output of this process resembles the FEM simulation data, in which a mesh is defined at the beginning and does not change along all the time-steps, and results are defined at each time-step.

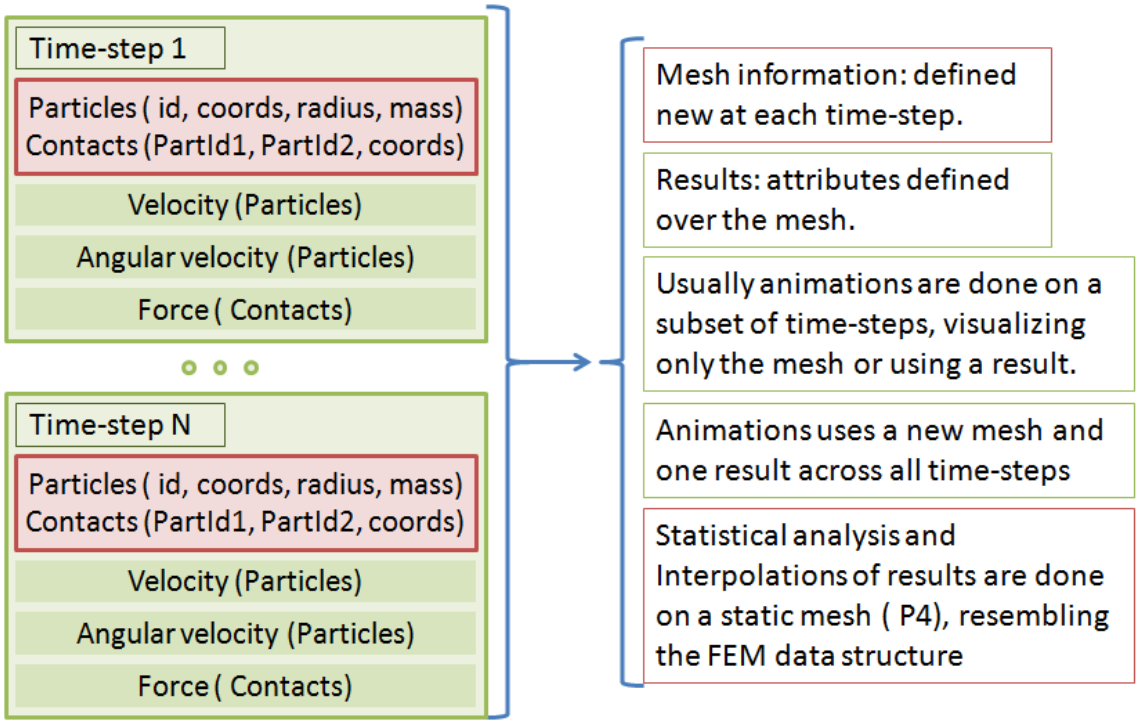

Figure 2.2 Simulation data structure of a DEM simulation run on a single machine.

Although Figure 2.2 represents particles as spheres, other particle types shall be supported but may be characterized as a group of spheres.

The provided DEM data is written using GiD ascii, binary format or Edinburg's P4 format, which is based on the one used in LAMMPS. Listing 2.3 shows an example of a P4 file for particles and Listing 2.4 for contacts. Note that different DEM codes output this type of information (particles and contacts) in formats different than P4 format.

```
TIMESTEP PARTICLES

0.1       1000

PID RADIUS MASS   PX   PY   PZ   VX   VY   VZ   WX   WY   WZ
1    0.1    0.2   0.0 0.0 0.0 0.1 0.1 0.1 0.1 0.1 0.1
...

1000 0.1    0.3   1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
```

```
TIMESTEP PARTICLES

0.2       1000

PID RADIUS MASS   PX   PY   PZ   VX   VY   VZ   WX   WY   WZ
1    0.1    0.2   0.0 0.0 0.0 0.1 0.1 0.1 0.1 0.1 0.1
...

1000 0.1    0.3   1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Listing 2.3 Sample P4 ASCII file format for particles.

```
TIMESTEP CONTACTS

 0.1       800

P1   P2   PX    PY    PZ    FX    FY    FZ
1    2    0.0   0.1   0.0   0.0   0.1   0.0
...

942  976  0.0   0.1   0.0   0.0   0.1   0.0


TIMESTEP CONTACTS

0.2       798

P1   P2   PX    PY    PZ    FX    FY    FZ
1    3    0.0   0.3   0.0   0.1   0.0   0.2
...

932  976  0.0   0.2   0.0   0.0   0.5   0.0
```

Listing 2.4 Sample P4 ASCII file format for contacts.

2.3 Distributed simulation data

When the simulation is executed in a distributed environment, such as an HPC cluster, then each node writes its own partition data, its own portion of the mesh and results, in its own file.

Following listings shows the nodes and elements of the telescope example for some of the partitions

```
# encoding utf-8
MESH "Kratos Tetrahedra3D4 Mesh" dimension 3
ElemType Tetrahedra Nnode 4
Coordinates
61466 8944.2969 2324.8501 2376.0071
62552 8946.8242 2325.7144 2377.313
…
4058312 8943.9102 2376.6377 2460.783
end coordinates

Elements
1096025 410088 3264881 3264883 3258074 0
```

```
# encoding utf-8
MESH "Kratos Tetrahedra3D4 Mesh" dimension 3
ElemType Tetrahedra Nnode 4
Coordinates
278669 8924.9619 2333.2236 2394.3804
287934 8926.8105 2333.0669 2396.5317
…
4062434 8879.8223 2368.1016 2441.2544
end coordinates

Elements
724816 2910964 2951322 2943183 370135 0
```

```
1096026 3264881 3264883 3258074 3258079 0
…
23662881 473256 3693657 3693658 2811908 0
end elements
```

```
GiD Post Results File 1.0

# encoding utf-8
Result "PART. INDEX" "Kratos" 21 Scalar OnNodes
ComponentNames "PARTITION INDEX"
Values
61466  31.0
62552  31.0
…
4058312  0.0
End values

Result "PRESSURE" "Kratos" 21 Scalar OnNodes
ComponentNames "PRESSURE"
Values
61466 -7.6108e-04
62552  1.3068e-02
…
4058312  4.9280e-02
End values

Result "VELOCITY" "Kratos" 21 Vector OnNodes
ComponentNames "X-VEL", "Y-VEL", "Z-VEL", "|V|"
Values
61466 -1.36e-01 -1.41e-01 2.07e-01 2.85e-01
62552 -1.56e-01 -1.62e-01 7.42e-02 2.37e-01
…
4058312  1.05e+00 -5.44e-02 2.41e-01 1.08e+00
End values

Result "PART. INDEX " "Kratos" 41 Scalar OnNodes
Values
…
End values

Result "PRESSURE" "Kratos" 41 Scalar OnNodes
OnNodes
Values
…
End values

Result "VELOCITY" "Kratos" 41 Vector OnNodes
Values
…
End values
…
```

Listing 2.3 data from Partition 0

```
724822 2910964 2951322 370135 2994026 0
…
23473884 4059187 530636 4059189 4059188 0
end elements
```

```
GiD Post Results File 1.0

# encoding utf-8
Result "PART. INDEX " "Kratos" 21 Scalar OnNodes
ComponentNames "PARTITION INDEX"
Values
278669  0.0
287934  0.0
…
4062434  1.0
End values

Result "PRESSURE" "Kratos" 21 Scalar OnNodes
ComponentNames "PRESSURE"
Values
278669  3.1364e-02
287934  2.5183e-02
…
4062434  4.2826e-02
End values

Result "VELOCITY" "Kratos" 21 Vector OnNodes
ComponentNames "X-VEL", "Y-VEL", "Z-VEL", "|V|"
Values
278669  5.35e-01 -8.79e-02 1.56e-01 5.64e-01
287934  6.26e-01 -7.50e-02 2.29e-01 6.71e-01
…
4062434  1.04e+00 -5.08e-02 2.13e-01 1.06e+00
End values

Result "PART. INDEX " "Kratos" 41 Scalar OnNodes
Values
…
End values

Result "PRESSURE" "Kratos" 41 Scalar OnNodes
OnNodes
Values
…
End values

Result "VELOCITY" "Kratos" 41 Vector OnNodes
Values
…
End values
…
```

Listing 2.4 data from Partition 1

```
# encoding utf-8
MESH "Kratos Tetrahedra3D4 Mesh" dimension 3
ElemType Tetrahedra Nnode 4
Coordinates
34549 8925.0723 2321.5095 2348.3384
35397 8925.7275 2326.3459 2347.1865
…
1153043 8907.4619 2331.3745 2357.1917
end coordinates

Elements
10825 67964 975985 975982 963807 0
10826 975985 975982 963807 963813 0
…
21966512 787659 813371 791758 43737 0
end elements
```

```
GiD Post Results File 1.0

# encoding utf-8
Result "PART. INDEX" "Kratos" 21 Scalar OnNodes
ComponentNames "PARTITION INDEX"
```

```
# encoding utf-8
MESH "Kratos Tetrahedra3D4 Mesh" dimension 3
ElemType Tetrahedra Nnode 4
Coordinates
236307 8910.8672 2333.1445 2381.342
236563 8910.3359 2333.0784 2381.0679
…
3270050 8904.6943 2341.377 2392.0667
end coordinates

Elements
1889 384923 3095737 3095740 3095741 0
1890 3095737 3095740 3095741 3103154 0
…
19147990 2632647 2626933 316102 2557512 0
end elements
```

```
GiD Post Results File 1.0

# encoding utf-8
Result "PART. INDEX" "Kratos" 21 Scalar OnNodes
ComponentNames "PARTITION INDEX"
```

```
Values
34549  112.0
35397  31.0
…
1153043  112
End values

Result "PRESSURE" "Kratos" 21 Scalar OnNodes
ComponentNames "PRESSURE"
Values
34549  1.4192e-03
35397 -4.8319e-05
…
1153043 -1.6497e-02
End values

Result "VELOCITY" "Kratos" 21 Vector OnNodes
ComponentNames "X-VEL", "Y-VEL", "Z-VEL", "|V|"
Values
34549  0.00e+00  0.00e+00 0.00e+00 0.00e+00
35397  0.00e+00  0.00e+00 0.00e+00 0.00e+00
…
1153043 -5.64e-02 -1.12e-01 1.68e-02 1.26e-01
End values

Result "PART. INDEX " "Kratos" 41 Scalar OnNodes
Values
…
End values

Result "PRESSURE" "Kratos" 41 Scalar OnNodes
OnNodes
Values
…
End values

Result "VELOCITY" "Kratos" 41 Vector OnNodes
Values
…
End values
…
```

Listing 2.5 data from Partition 119

```
Values
236307  127.0
236563  127.0
…
3270050  92.0
End values

Result "PRESSURE" "Kratos" 21 Scalar OnNodes
ComponentNames "PRESSURE"
Values
236307 -1.8706e-03
236563 -3.2198e-03
…
3270050  9.9210e-03
End values

Result "VELOCITY" "Kratos" 21 Vector OnNodes
ComponentNames "X-VEL", "Y-VEL", "Z-VEL", "|V|"
Values
236307 -1.06e-01 -1.58e-02 1.63e-01 1.95e-01
236563 -1.08e-01 -1.84e-02 1.29e-01 1.69e-01
…
3270050  7.72e-01 -8.71e-02 2.05e-01 8.03e-01
End values

Result "PART. INDEX " "Kratos" 41 Scalar OnNodes
Values
…
End values

Result "PRESSURE" "Kratos" 41 Scalar OnNodes
OnNodes
Values
…
End values

Result "VELOCITY" "Kratos" 41 Vector OnNodes
Values
…
End values
…
```

Listing 2.6 data from Partition 127

As can be seen each partition has a non-contiguous subset of the nodes and elements of the whole mesh. And nodes at the interface between partitions, i.e. shared by elements of two partitions, are duplicated.

To get the final whole model, all partitions are merged:

- using the mesh names and properties, such as element type, all mesh parts are glued together to visualize the whole domain and
- using the result name and properties, such as result type and time-step value, each time a result is accessed, all 128 parts are read and merged together.

2.4 Data localization

The HPC clusters where simulation programs run are composed of a set of powerful compute nodes with minimal local storage and are themselves connected to each other using a high-speed network such as Infiniband. The result output of the simulation is written on a centralized, high efficient file system like Lustre, or NFS.

In the case of the telescope example, the central NFS file system contains all 128 files corresponding to the 128 sub-domains.

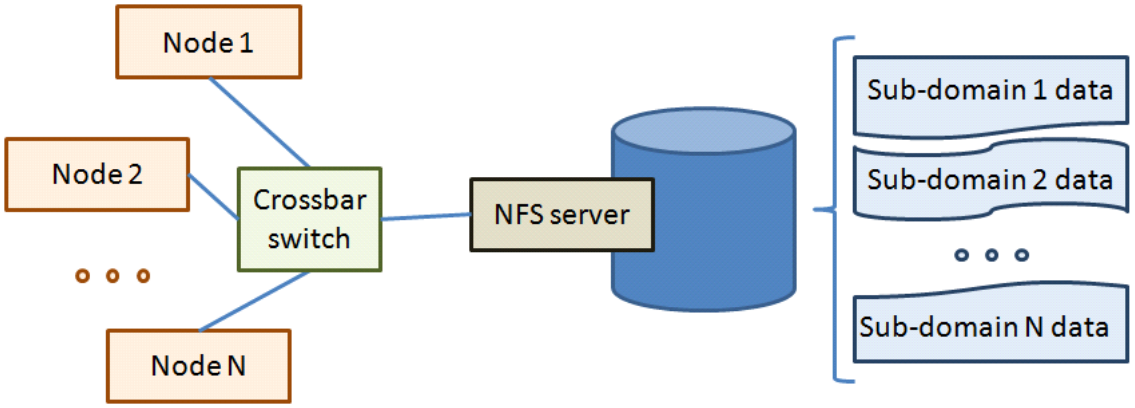

Figure 2.3 Localization of the simulation data

2.5 Target data characteristics

Table 2.1 shows for a single simulated model, the sizes of the data to be handled by the system: size of the mesh, number of attributes per node or particle, number of expected time-steps, and number of sub-domains for the single simulation.

	DEM	FEM
Total size of the data for the simulated model	From 50 Gigabytes to Petabyte	From 30-140 Gigabytes to 12-50 Terabytes
Number of partitions	1 to 10,000	1 to 10,000
Number of particles / elements	10 million particles	From 8-100 millions to 1 billion tetrahedrons
Number of written time steps	1 billion	From 40 to 25,000
Number of variables per particle / node	Particles: 12 (3 scalars + 2 vectors) + user defined variables (scalars and vectors) Contacts: 8 (2 scalars + 2 vectors + user defined variables (scalars and vectors)	6 (2 scalars + 1 vector) to 16 (8 scalars + 2 vectors)

Table 2.1 Characteristics of a single simulation to be handled by the VELaSSCo platform

3 VELaSSCo platform overview

3.1 Interaction

The main use case of the VELaSSCo platform from the user's point of view may be as follows. The user requirement evaluation with the User Panel that is underway will provide critical input into defining these.

1. The **user** wants to visualize a result from an already executed simulation on the HPC cluster.
2. The **user** starts a visualization client, GiD, in its own computer and tells the client to connect to the HPC cluster.
3. The **visualization client** connects to the VELaSSCo platform installed in the HPC cluster.
4. The **VELaSSCo platform** verifies the user's identity.
5. The **VELaSSCo platform** returns a list of the available simulations data with a summary (date, size, etc.), which the client shows to the user.
6. The **user** selects one of the models, selection forwarded by the client to the platform.
7. The **platform** returns a view, mesh, of the model (skin extracted from the model), eventually with the last visualized result.
8. The **user** interacts with the model: rotates, zooms, and switches layers on/off, changes styles like wireframe, solid view, among others.
9. The **user** issues a query, for instance the iso-surface of a result of a certain time-step, and the **visualization client** forwards the query to the platform. Several queries can be performed and are detailed further down.
10. The **platform** analyzes the query, accesses the simulation data and performs the calculation of the iso-surface and returns the iso-surface mesh to the **visualization client** which shows it to the user.
11. Steps 8 through 10 are repeated until the user decides to open another model, and then go to step 5, or close the session.

Depending on the query performed in steps 9 and 10, more interaction may be needed or feed-back should be provided during its execution.

Following diagrams give an overview of the behaviour or the system. Figure 3.1 shows an overview of the processes launched by the user and involved in the feeding of data to the system. Figure 3.2 shows the different components of the whole system and their relations. Figure 3.3 displays the interaction between the user, the client and server parts of the system.

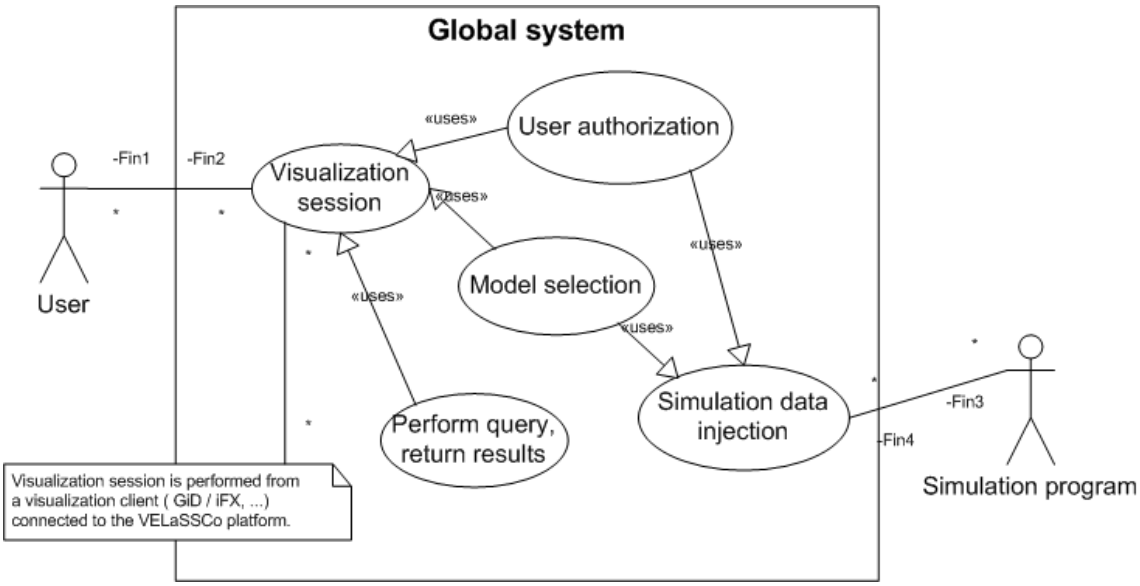

Figure 3.1 Global overview of the VELaSSCo platform

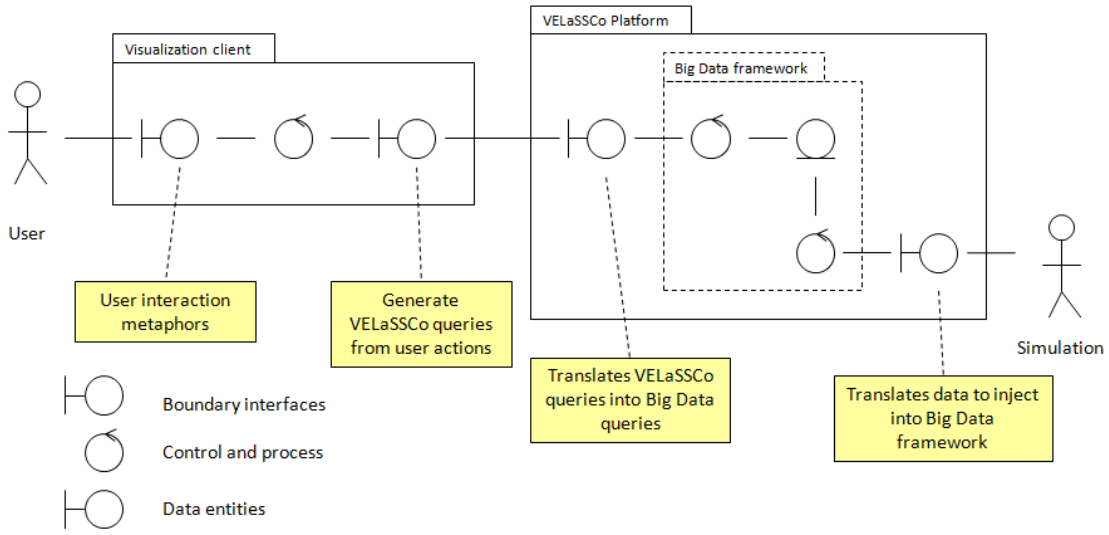

Figure 3.2 Components of the global system

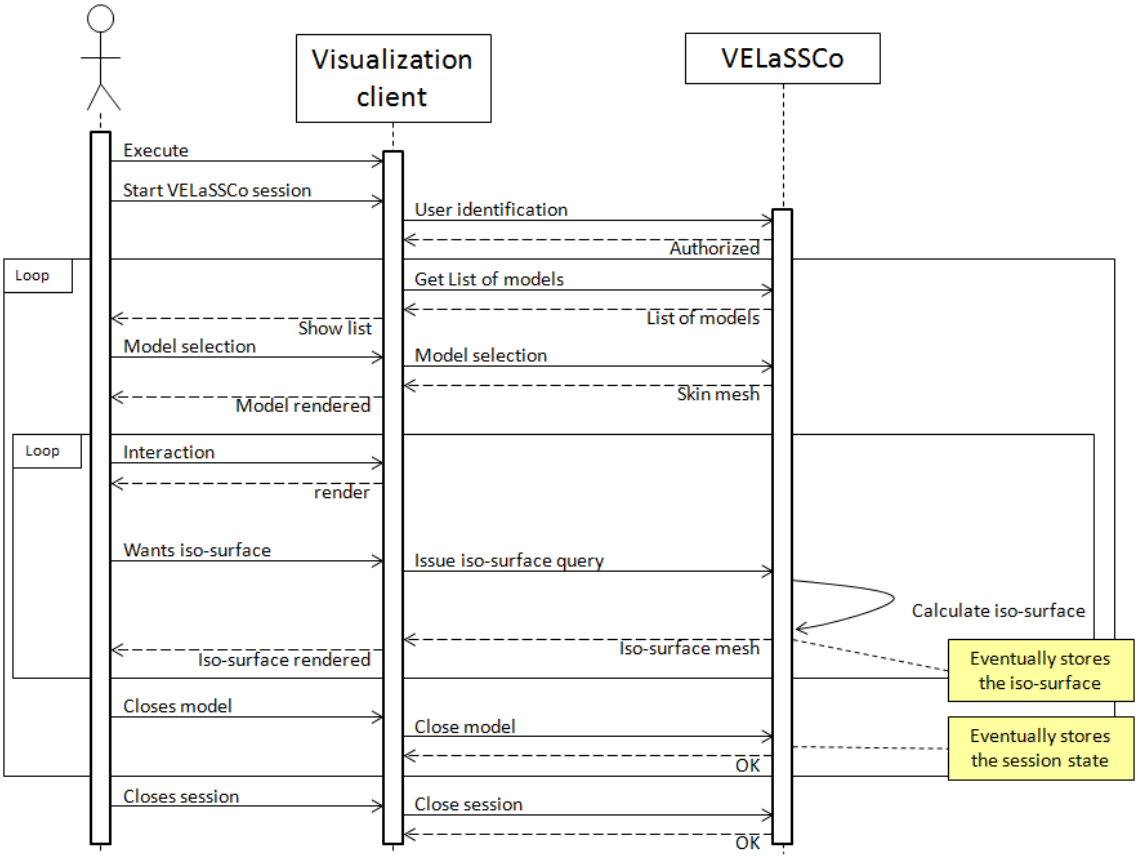

Figure 3.3 Sequence diagram of user interaction with the system.

The "Simulation data injection" process represents the eventual transformation and transfer of the simulation data from the current format to the one usable by the system. It should be evaluated if the injection process should also merge together the sub-domains of the calculation process so that, for instance, the mesh data can be handled as a single data set is one instead of the N data-sets, one for each sub-domain, and which the map-reduce process should merge.

Depending on the big data solution selected for the VELaSSCo platform, the simulation data can be accessed directly or it should be injected into the VELaSSCo platform.

A mechanism should be provided so that a running simulation, instead of writing the results on a file, is able to inject the “result time-step slice” directly to the platform using a standard open format.

3.2 Ensuring interaction

A simplification module, multi-resolution model, is also to be incorporated to the platform in order to ensure the interaction between the user and the visualized model and results.

This process and other processes involved in the queries, such as the discrete to continuum interpolation, are incorporated in Figure 3.4, a refinement of Figure 3.2.

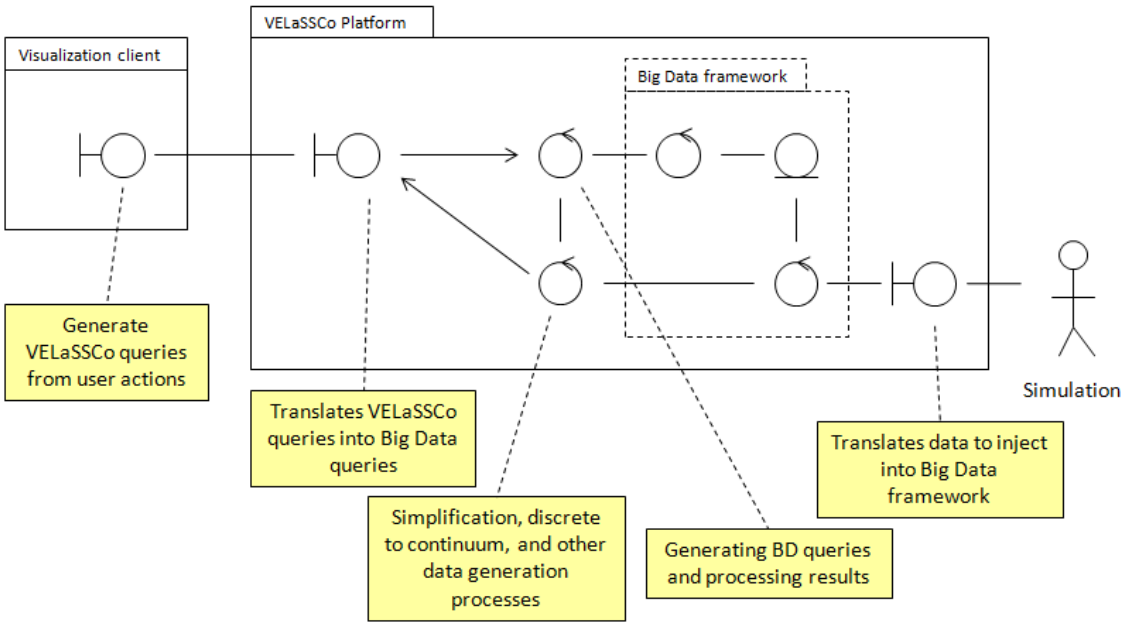

Figure 3.4 Incorporating components which generate new data for the system.

3.3 Queries

3.3.1 User queries & visualization client queries

When the user interacts with the model and wants to see an iso-surface or display a contour fill of a result, then the visualization client sends a query to the visualization platform like these ones:

- **query**(session_id, *create_iso-surface*, result_identification) returns the mesh of the iso-surface which is rendered by the client;
- **query**(session_id, *get_results_values*, result_identification, list_of_nodes) returns a list of result values for the provided list of nodes. The values are then used by the visualization client to draw a contour fill, or contour lines, or vectors, over the mesh the visualization client already has;
- result_identification can be done with: (result_name, step_number, analysis_name).

Several user functional requirements can be translated into a single query between the visualization client and the VELaSSCo platform. For instance:

- getting a result value for node and draw it as label,
- getting the result values for some, or all, time-steps for a certain node for a node evolution graph,
- getting the result values to draw a contour fill, contour lines, vectors over the skin of the model;
- can be grouped into: query(session_id, *get_result_values*, result_name, list_of_time_steps, analysis_name, list_of_nodes) which returns, for each time-step of the list, a list of result values for the provided node list.

From the user functional requirement list, a table should be created by grouping similar queries into fewer queries to be issued against the VELaSSCo platform. Also the implications of the queries should be taken into account, for instance getting the result values for the nodes of a cut plane implies that those values are to be interpolated from the original model.

3.3.2 Processing visualization client queries

The queries issued by the visualization client are processed by the VELaSSCo platform and can result in one or more internal queries.

An interface will be needed to translate the queries received by the visualization client into the internal processes to calculate and provide the results.

As example, if the solution selected for the VELaSSCo platform is Hadoop with MapReduce, then the iso-surface query can be translated into a MapReduce query:

- **map**: get the iso-surface for each portion of the mesh,
- **reduce**: merge the different iso-surface chunks into a single isosurface mesh,
- **transform**: the iso-surface mesh is translated into a fault-tolerant, GPU friendly format and sent to the visualization client.

Staying in the same framework example, a stream-line calculation can generate multiple MapReduce queries:

- **map**: from the seed point list, generate stream-lines at each portion of the mesh,
- **reduce**: check if the stream line finishes at a boundary or in a vortex, if not, then new seed points (at the boundary between partitions) are added to the seed list,
- **loop**: if there are still seed points on the list issue a new MapReduce query until the seed list is empty.

The translation from the visualization client into internal ones depends on the big data architecture selected for the VELaSSCo platform.

3.3.3 Grouping user queries into visualization client queries

Visualization of:

- mesh:
  - from original model: skin extraction, layer selection, single surfaces
  - from DEM original model: particles and contacts (a coarsening model may be needed if original information hinders user's interactivity)
  - created by the user (thus information must be interpolated from original data): cut planes, cut lines, iso-surfaces, points
- results:
  - contour fill, contour lines, vectors over original mesh → needs result values on a list of nodes
  - contour fill, contour lines, vectors over created mesh → needs interpolation
  - graphs: point queries, line queries → may need interpolation
  - statistical results: process some or all original data, an example of algorithms calculating statistical results over particles are present in P4 tool and may be incorporated in the VELaSSCo platform.

The interpolation of result from the original mesh to a cut-plane, cut-line or iso-surface can be done at two places:

- inside the VELaSSCo platform: needs the information of the cut-plane, cut-line, iso-surface to access the affected elements and interpolate the values,
- in the visualization client: then when a cut-plane, cut-line or iso-surface is created the visualization client also needs the list of affected elements so that when the results are needed on these surfaces, the client can ask the platform for the values on those elements and interpolate the values

There are two types of queries:

- consulting queries: get information from original mesh, from original results,
- object creation / interpolation queries:
  - over the mesh: skin extraction, cut-planes, cut-lines,
  - over mesh and results: iso-surfaces, stream-lines, interpolation over created mesh objects (skin extraction, cut-planes, cut-lines) if interpolation is done in VELaSSCo,
  - over the results: creation of statistical results from existing ones.

→ a decision should be taken on where the interpolation should be done: whether in the visualization client or in the VELaSSCo platform.

→ a decision should be taken where to store the created meshes (skin, iso-surfaces, cut-planes, cut-lines, etc.)

→ multi-resolution model, simplified meshes to ensure interactive rates should be stored in the platform

→ Discrete to continuum transformation should also be stored in the Big Data framework.

If the created objects (cut-planes, iso-surfaces, etc.) are stored in the big data framework, some kind of object identifier is needed in order to interpolate results over these created objects or query other information.

3.3.3.1 Session queries

QS1: User → client: start session (login)

- client → platform: start session (login)
- platform → client: validation

QS2: User → client: get list of models (user initiated or client initiated)

- client → platform: get list of available models
- platform → client: list of available models (with summary information: global mesh & results size, partitions, etc.), for instance number of total nodes, number of total elements, number of analysis, time-steps and results per time-step.

QS3: User → client: model selection

- client → platform: model selection
- platform → client: ok

QS4: User → client: close model

- client → platform: close model
- platform → client: ok

QS5: User → client: end session

- client → platform: end session
- platform → client: ok

3.3.3.2 Mesh queries

QM1: User → client: get model information (user initiated or client initiated)

- client → platform: get model information
- platform → client: summary information: list of meshes/layers with their mesh information, results size,

QM2: User → client: visualization of model mesh

- client → platform: get boundary, skin, meshes
- platform → client:
  - *FEM & DEM continuum*: calculates and returns the (simplified) boundary meshes for the volume meshes and all other meshes (surfaces, line or points) of the model **(is the same for all time-steps of the analysis)**,
  - *DEM particles and contacts*: return (simplified) mesh of spheres (or lump of spheres for non-spherical particles) and surface, line (contacts) or point meshes **over selected time-step**.

Simplification: new meshes are created from the original ones.

QM3: User → client: creation of cut-planes, cut-lines, cut-segments or "point-sensors"

- client → platform: get mesh for cut-plane, cut-lines, cut-segments or "point-sensors"
- platform → client:
  - *FEM & DEM continuum*: calculates and returns the (simplified) meshes for the defined cut-planes, cut-lines, cut-segments or point-sensors,
  - *DEM particles and contacts*: (over selected time-step or over all time-steps) calculates and returns the (simplified) meshes for the defined cut-planes, cut-lines or point-sensors,

New meshes are created from the original ones, and simplified.

QM4: User → client: visualization of model boundary conditions (for the simulation) and parameters

- client → platform: get boundary conditions and parameters,
- platform → client: returns boundary conditions and parameters,

QM5: User → client: create/visualize the continuum mesh from the DEM particle simulation

- client → platform: create continuum mesh from DEM particle data (needs parameters: averaging, etc.),
- platform → client: calculates and returns summary of the continuum mesh
- client → platform: get boundary, skin, meshes (as in **QM2**)
- platform → client: *FEM & DEM continuum*: calculates and returns the (simplified) boundary meshes for the volume meshes and all other meshes (surfaces, line or points) of the model **(is the same for all time-steps of the analysis)**,

May be both client → platform queries can be merged into one.

May be this query is to be performed together with query **QR7,** create temporal and spatial averaging of the selected results, as described on DEM-discrete data User requirements (Visualisation and DEM analysis) in Deliverable D1.1

QM6: User → client: visualization of complex particles

- client → platform: get complex particles for the actual time-step,
- platform → client: complex particles definition, information, for the actual time-step,

QM7: User → client: perform a 'travelling' through the model (interactive navigation)

- client → platform: get mesh and result information of current region of interest,
- platform → client: returns the mesh and result for the current ROI,
- **loop:** as the user travels along a path (modes the ROI), the **client** asks for updates on the new ROI, the **platform** sends the new information that enters the ROI.

Depending on the design of the platform following queries may be needed:

QM8: User → client: interactive visualization

- client → platform: get simplified skin, surface and spheres mesh of model,
- platform → client: calculates and returns the simplified meshes.

QM9: User → client: detailed full-resolution of Region of Interest (zoom area)

- client → platform: get original skin, surface and spheres mesh of model, with current visualized results,
- platform → client: returns the full-resolution meshes section, with the selected results.

3.3.3.3 Result queries

QR1: User → client: visualize contour fill, contour lines, vectors, …

- client → platform: get result values for list of nodes (of skin or other surface meshes) for the actual time-step
- platform → client: returns a list of the results values from the selected time-step,

Vectors may be represented on nodes of the volume mesh, may be a special query **QR1b** is required.

As explained in Section 3.2.3 a decision should be taken on where the interpolation for **QM3**, **QR2** and **QR3** should be done: whether in the visualization client or in the VELaSSCo platform.

QR2: User → client: visualize iso-surface

- client → platform: create and get iso-surface (requires criteria: result identification and result value parameters)
- platform → client: calculates and returns the (simplified) surface meshes for the iso-surface on the volume meshes and the line-meshes for the iso-lines on the surface meshes

  New meshes are created from the original ones, and simplified.

QR3: User → client: visualize stream-lines

- client → platform: get stream-lines for the provided seed points.
- platform → client: calculates and returns the (simplified) line mesh of the stream-lines defined by the user seed points,

This query can be seen as a combination of **QM3** and **QR1**, as explained in Section 3.2.2.

As explained in Section 3.2.3 a decision should be taken on where the interpolation for **QM3, QR2** and **QR3** should be done: whether in the visualization client or in the VELaSSCo platform.

QR4: User → client: animate any of the above result visualizations (queries)

- client → platform: perform the same queries but updating the time-step value
- platform → client: eventually calculates and returns the information desired for the selected time-step

QR5: User → client: calculate statistics (minimum, maximum, average, standard deviation, etc.) for the selected result.

- client → platform: calculate statistical results for the user selected result.
- platform → client: calculate statistical results for the user selected result and returns a summary

This query will be followed by a visualization query, like **QR1**.

QR6: User → client: create temporal and spatial averaging of the selected results

- client → platform: calculate temporal and spatial averaging of the DEM particle mesh,
- platform → client: calculate temporal and spatial averaging of the DEM particle mesh and returns a summary

This query will be followed by a visualization query, like **QR1**.

May be this query is to be performed together with query QM5, create/visualize the continuum mesh from the DEM particle simulation, as described in Section 2.1.2 of Deliverable D1.1.

3.4 Interfaces

3.4.1 User ←→ Visualization client

The interaction between the user and the visualization client should be (almost) the same as if using a "heavy" model locally. So the connection to the VELaSSCo platform should be (almost) transparent, only evident for the extra functionalities in the visualization client such as:

- connection and user identification on the VELaSSCo platform,
- model summary and availability listing and selection,
- eventually provide feed-back for complex and heavy queries.

To ensure interaction with the model, a simplified representation will be used, but full-resolution information will be at users disposal.

3.4.2 Visualization client ←→ VELaSSCo

The communication between the visualization client and the VELaSSCo platform should be efficient, secure, fast, robust and reliable.

The information flow between the VELaSSCo platform and the visualization client should be done in such way that if some parts of the query results are lost, still some visualization can be performed. The missing parts should be detected and send again.

For instance if the platform sends back the mesh and results of an iso-surface, and some chunks of the information are lost, the iso-surface should still be able to be rendered, but with holes representing the missing information, which then will be gradually filled.

3.4.2.1 Combined post-processing

To ensure interactiveness and provide a faster feedback for end users viewing parts of the simulation results (at full resolution) or the whole model (coarser resolution), part of the post-process may be performed in the visualization client and not only on the HPC cluster. One example would be to allow the interactive generation of streamlines using parallel processing units (CPU/GPU), which would enable the user to control seeding of particles in real-time. Another example would be the display of volume data. Instead of generating iso-surfaces as polygons with the help of, e.g., a marching cubes algorithm, iso-surfaces can also be visualized using ray casting algorithms (also running on multiple CPUs/GPUs). This makes it possible to change iso values on the fly and display new surfaces without having to run a geometric conversion algorithm. Creating iso-surfaces, stream-lines or other post-process steps on the visualization client may require a coarser representation of the simulated data when performed on the whole model, as the full resolution data cannot be handled by the visualization client capabilities.

Running the post-process queries on the visualization client on partial simulation data, or on a coarser model can also be seen as an opening step before the user issues the queries remotely on the full resolution model.

3.4.2.2 Progressive display

Another class of visualization use-cases comprises progressive display scenario, where data is continuously streamed from the HPC cluster to the visualization machine. This realizes a typical query based visualization (QBV) setting, where results are sent on request with low latency. One example is progressive level-of-detail, where more geometric detail is added on demand based on a multi-resolution model. Another example would be interactive navigation through a (spatially) large dataset, where new data is sent to the visualization machine as soon as the user moves to a different part of the model. Another example is the animation of a result, where at each time-step the user selected result is sent to the visualization machine to create the appropriate image for the animation.

3.4.2.3 Requirements

In summary, the visualization requirements are represented in Table 3.1

	Combined post-processing	Separated post-processing
Progressive display	Post-processing, and rendering, is done on the visualization machine. The HPC cluster continuously sends progressive visualization data.	Post-processing is done on the HPC cluster. The visualization machine only displays geometry. The HPC cluster continuously sends graphical primitives.
Non-progressive display	Post-processing, and rendering, is done on the visualization machine. Data is sent as a large chunk, which is used for some time until a new chunk is requested.	Post-processing is done on the HPC cluster. The visualization machine only displays geometry. Data is sent as a large chunk, which is used for some time until a new chunk is requested.

Table 3.1 Classification of requirements according to the visualization use-cases.

Depending on the use case, there are different requirements for the **input data** that is sent by the HPC cluster to the visualization machine:

- **Combined post-processing**. If post-processing is done on the visualization machine, we would have to deal with a wide variety of simulation results. A possible data format would have to support arbitrary sets of spatial cells (tetrahedra, pyramids, cubes, etc.), different kinds of connectivity (structured, unstructured data), and arbitrary attributes associated with cells as well as vertices.

- **Separated post-processing**. If the visualization machine performs only display of geometric data, a suitable data format would have to support multiple graphics primitives, such as polygons, NURBS, etc. with attributes defined on them.
- **Progressive Display**. This is subject to research and requirements and structure of a data format supporting progressive display is yet to be determined. Since this setting is targeted towards high interactivity, a minimum requirement is that transferred data has to be as light-weight as possible to ensure high transfer rates. Ideally, this should be binary data with only a minimum of control structures. In the best case, such data would be loaded into the address space of the RPE CPU process (and/or a GPU kernel) and directly used without any parsing or conversion.
- **Non-Progressive Display**. In this case we would assume that all data is converted by RPE into an internal binary representation. Here, data could be arbitrarily formatted and an ASCII representation would be acceptable.

Performance should always be taken into consideration, and overhead in a chosen data format should be minimized. However, in non-progressive settings, almost any format that supports all needed data types should be acceptable.

In a progressive setting, the data format certainly would be highly specialized, and there is probably no **open** format available that would fit all our needs.

The queries specified in section 3.2 may fall into one or several of the above four use-cases.

3.4.3 VELaSSCo ←→ Simulated data

The versatile mechanism to incorporate data into the big data framework should target not only handle already existing simulation data but be able to interact with running simulations to gather the results at the precise moment they are being calculated.

3.5 Context

As described on section FEM-Continuum data User requirements (Context) in deliverable, the platform should be able to co-exists with existing HPC cluster and resource management. The HPC clusters used by CIMNE are 64-bit Linux system with several powerful nodes with minimal local storage and connected with their selves using a high-speed network such as Infiniband. Usually storage is centralized using Lustre or NFS. The typical resource managers and job schedulers are Slurm, Torque or Sun grid engine.