Patty / Mapping the Via Appia in 4D

Via Appia 4D GIS: Software User Manual

E. Ranguelova
R. Goncalves
R. van Haren
O. Martinez-Rubi
Netherlands eScience Center,
Science Park 140 (Matrix 1), 1098 XG Amsterdam, the Netherlands

May 22, 2015

Contents

1 Overview

The subject of this project is an archaeological study of an area full of funerary monuments between the fifth and sixth miles of the Via Appia Antica (`http://en.wikipedia.org/wiki/Appian_Way`). The data gathered from this area is of different resolutions and types: point clouds, meshes, photos, historical images, 2D footprints and attribute information of the monuments. The data is managed, processed and visualized by the Via Appia 4D Geospatial Information Systems (GIS). Such system combines a Windows desktop viewer, a web-based viewer and a Database Management System (DBMS).

Through a client-server architecture, multiple researchers at different locations are able to analyze and study different areas of the ViaAppia. Such architecture creates the base for 4D GIS systems for the exploration of large and complex archaeological areas.

This document is the user manual of the developed Via Appia 4D GIS system. The remainder of it is organized as follows: the data are heterogeneous in nature due to both their conceptual difference and acquisition method. The conceptual data description can be found in Section 2. The client-server architecture of the system is described in Section 3. The storage of the data is organized according to a predefined hierarchical structure as described in Section 4. The database storing information about the location of the data as well as many attribute data (meta-data) of archaeological interest is explained in Section 5. The software used in the 4D GIS is described in Section 6. The report finalizes with an overview of future steps to be taken in Section 7

2 Conceptual description

An area between the fifth and sixth miles of the Via Appia Antica in Rome is studied and different types of data are collected:

- *Point clouds (PC)*
  - A low resolution PC of the studied area that has been generated making use of Fugro's DRIVE-MAP services.
  - High resolution PCs of the different monuments generated using photogrammetric technologies.
- *Meshes* reconstructions (3D models) of the monuments for different historical epochs.
- Contemporary and historical *pictures* or paintings.
- Footprints in 2D of the monuments acquired using Topcon positioning systems.
- *Attributes* data for the monuments and their parts, which are gathered by field observations.

We conceptually divide the area into items and there are two types of items:

- *background*: It is the whole area considered as one single entity (item).
- *sites*: The different monuments of interest contained in the area. They are considered as different entities (items)

The point clouds, meshes and pictures are stored according to the data structure defined in section 4. In the next subsections we provide more information of the different types of data and we specify to which type of item they are related according to the previous conceptual definition.

2.1 Point clouds

Different types of point clouds are collected and handled

2.1.1 DRIVE-MAP

Using the DRIVE-MAP service of Fugro the the road was scanned and a *PC* was produced. A point in the PC has 3D coordinates (x, y, z), in respect to a given Earth reference system known at scanning time, and it also has color and other attributes as classification. The resolution (number of points per area or volume) of this point cloud is not enough to see detailed features of the sites. Furthermore, they were only scanned from the main road. Thus, the back of the sites is missing. This is illustrated on Fig.1.

Figure 1: Point cloud data gathered along Via Appia. The Drive map is of low resolution and covers only part of the monuments (sites) facing the road.

There are different versions of the same DRIVE-MAP, each version contains different processed point clouds. For example, there is a version where the points from some trees have been removed (using the classification attribute of the points).

The DRIVE-MAP dataset is *background* data.

2.1.2 Photogrammetry

Additional point clouds of higher resolution covering also the back of the monuments have been obtained using photogrammetric software from photographic pictures of the sites from different camera locations (viewpoints). Such point clouds are *sites* data.

2.2 Pictures

There are several pictures of the sites, both contemporary (current) and historical pictures or paintings. Fig.2 has an illustration of visualizing a contemporary photo of a monument next to its point cloud data. The photo was one of the photos used to generate the high resolution site PC and visualizing it next to the location of the monument provides extra information about the context. Each site has multiple pictures. The sites pictures are *sites* data.

2.3 Meshes

For the purpose of archaeological research, multiple reconstructions of the sites of interest are often performed. These reconstructions, or meshes, are also classified as current or historical. An illustration of visualizing a contemporary mesh of a monument aligned to its point cloud data is pictured on Fig.2. The different meshes are *sites* data.

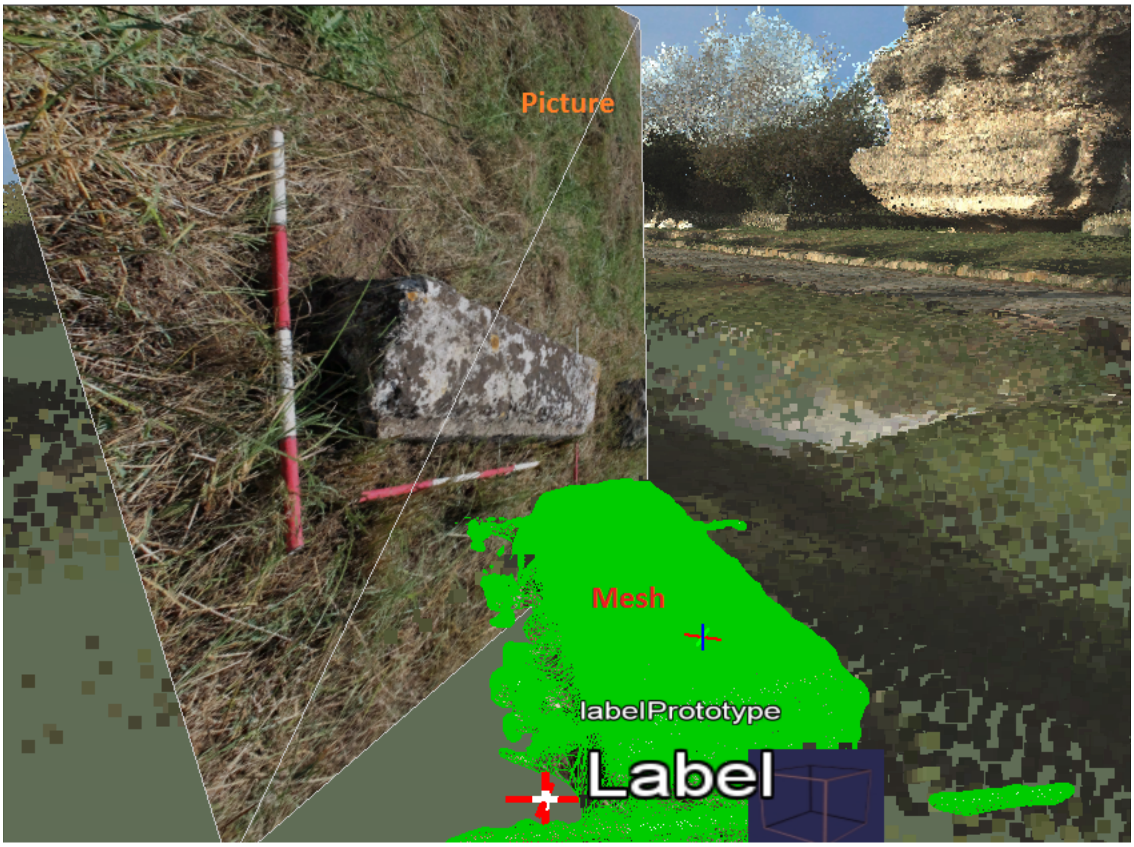

Figure 2: Reconstruction meshes and picture of a site overlayed on the point cloud data.

2.4 Footprints

Topcon 2D footprints were also collected for the different sites in the area. This data is georeferenced. This is *sites* data.

2.5 Attributes

Attributes represents the data collected at the site by the archaeologists, for example, material composition, condition, possible interpretation, description of the different elements or subparts, etc. The attribute data is considered as additional descriptive (meta) data and it is stored in a database along with the pointers to the other data types (Section 5). Furthermore, they are extremely relevant for the definition of archaeological research questions. This is *sites* data.

3 System architecture

The developed 4D GIS has a two-tier *client-server* architecture. The *server* contains the master copy of all the involved data and a PostgreSQL database called *viaappiadb* to ease the data handling. Both are used for data preparation, i.e. converting the data to the required formats for the available viewers. These run in the laptops of the archaeological researchers (*clients*) where local copies of the converted data in the *server* are downloaded when required.

3.1 Data preparation at server

A diagram of the data preparation procedure which is carried out in the *server* is shown in Figure 3. As explained in section 2) for the whole area, conceptually referred as background, we have only point cloud data, while for the individual monuments, conceptually referred as sites, the data is of more types (point clouds, meshes, pictures, 2D footprints and attributes).

In order to run the Windows desktop viewer in the clients the raw data has to be converted to the OpenSceneGraph (`http://www.openscenegraph.org`) binary format. The web-based viewer is using the POTree (`http://potree.org/`) WebGL point cloud visualization framework and the data needs to be converted with the POTree Converter which restructures the data into a OctTree where each leaf node is a file of format LAS or LAZ.

During the data preparation process the *viaappiadb* database is filled with meta-data information which contains the raw data location and converted data location. Furthermore, the archaeological information with attribute data for the several sites is extracted from a Microsoft Access file and imported into the *viaappiadb*. The footprints are provided in a ShapeFile and they are also imported into the *viaappiadb* database as well. In addition, altitude ranges for the sites are derived from the PC raw data in combination with the footprints and they are also added to the database. On Figures 3 and 4 the solid arrows indicate data flow, while the dashed arrows indicate meta-data flow during the data preparation process.

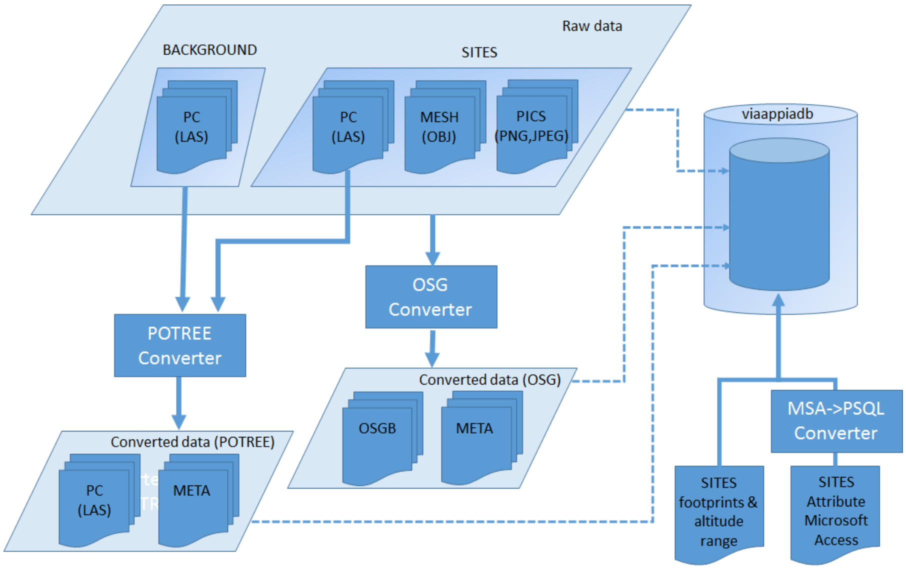

Figure 3: Data preparation framework executed on the Via Appia Linux serer

3.2 Visualization at clients

Once the data preparation framework has concluded its task the data is made available for multi-client sessions. In the case of Windows desktop viewers a local copy of all the OSG data is required before initializing the viewer.

A *launcher* tool based on the Xenon library (`http://nlesc.github.io/Xenon/`), developed by the Netherlands eScience Center (NLeSC), makes sure the local copy is synchronized with the server. It also retrieves from the server the configuration file required by the desktop viewer. Once the synchronization finishes the launcher is in charge of starting the viewer. In the end of a session the launcher updates the remote database using the modified configuration file. Offline operational mode is also provided, this is, the user decides when to synchronize and takes care of it.

For the Web-based viewer it is not required a local copy of the entire Potree converted data. The necessary data is pulled automatically by the web application from the server on request via a NGINX Web server (`nginx.com`). Such data also includes a JSON configuration file containing the meta-data for the background and sites information extracted from the *viaappia* database. The 3D Web-based viewer have been developed by NLeSC. Figure 4 illustrates the two-tier architecture and shows the steps performed at the client side.

All the process described so far has been automated. Section 6 contains more detailed information about the architecture.

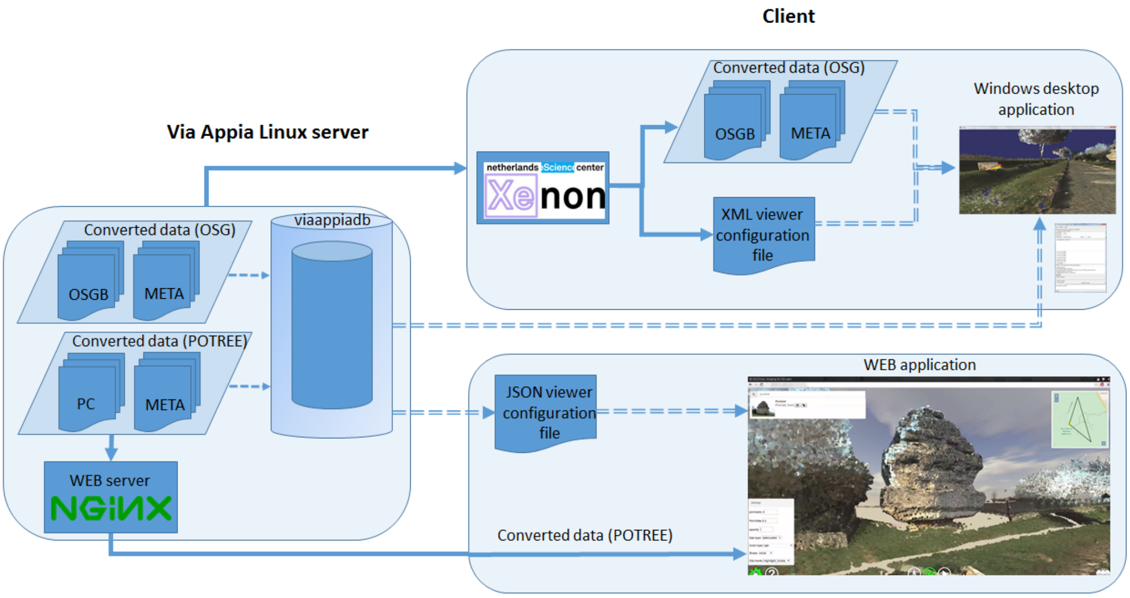

Figure 4: Two-tier architecture of the Via Appia 4D GIS and the steps executed on the client side.

4 File Storage

The file storage is organized into three major directories, *RAW*, *OSG* and *POTREE*. Their structure and the nomenclature is described in the coming sections as well as the scripts to manipulate their content.

It is important to mention that all the operations that modify the data must be done with *pattydat* user.

4.1 File storage structure

The raw data is **only** stored in the Via Appia Linux server. Through a python script, called *GenerateOSG.py*, raw data is converted to OSG data. Another script, called *GeneratePOTree.py*, converts the raw data into POTree internal format. More details about these two scripts can be found in Section 4.2.4 and Section 4.2.5.

These two scripts pick raw data from */home/pattydat/DATA/RAW*. After the data generation, the OSG data is stored in */home/pattydat/DATA/OSG* and POTREE data is stored in */home/pattydat/DATA/POTREE*, c.f., Figure 5.

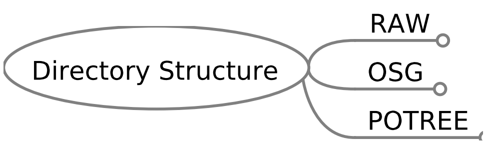

Figure 5: Overview of the data structure

4.1.1 Nomenclature

The directories and files name follow a specific nomenclature which describes a bit their content. Such information is then used by the data preparation scripts to locate the required data and also by the system to serve the user requests. These rules apply to all directories and files under RAW, OSG and POTREE data directories.

Under the RAW data directory there are sub-directories for point clouds, meshes and pictures. Each of them have a sub-directory for backgrounds called *BACK* and one for sites called *SITE*. For backgrounds, the subdirectory contains different folders with point clouds for each background. For sites, the subdirectory contains a separate folder for each site. For example, *MESH/SITE/CURR/S162* could contain two folders called *162_curr_1* and *162_curr_2*.

Point clouds and meshes can have two types of background and sites: (a) current representations, and (b) archeological reconstructions. They are stored in respectively *CURR* and *ARCH_REC*. For pictures the sub-division has a different nomenclature. The subdivision is made between (a) pictures of the current state of the sites, and (b) historical pictures and paintings. These are stored in respectively *CURR* and *HIST*.

For the point clouds the LAS files contained in each site sub-folder may have been prealigned through a third party tool such as CloudCompare. In that case the LAS file name must contain ***_ALIGNED_*BGNAME**** where *BGNAME* is the background name (as contained in the folder *PC/BACK/*).

Some point clouds generation tools store the color information in 8 bits instead of the usual 16 bits. In that case the folder name must be ***_8BC**. The effect of having an undeclared LAS file with 8 bit color is that the converted data will be black and white. Note that these properties are cumulative, for example *S162_ALIGNED_DRIVE_1_V3_8BC* is a valid name for a folder containing a LAS file with 8 bit color information and aligned points.

This hierarchy of folders and nomenclature for the raw data, as shown in Figure 6, is also used for the OSG directories, Figure 7, and for the Potree directories, Figure 8.

Note that the defined data structure also contemplate the possibility to have meshes and pictures for *background* even though these are not currently used.

4.2 Raw data management

On the data structure it is possible to perform different operations. Such operations are (1) adding raw data items (Section 4.2.1); (2) removing raw data items (Section 4.2.2); and (3) listing raw data items (Section 4.2.3).

4.2.1 Adding raw data items

The script *AddRawDataItem.py* is used to add raw data items to the directory structure and must be rub with *pattydat* user. To run the script, the user needs, depending on the exact datatype, to supply additional arguments to the script. An overview of the possible arguments are found in the verbatim below. The script automatically creates a destination folder with the required naming, as specified in section 4.1, using the supplied arguments.

For PC of sites the (*-f, --file*) argument should be the path to the LAS/LAZ file. For background PC it should be the folder. For PICT it should be the path to the png/jpg/jpeg file but it is also possible to specify a folder with several pictures (but they should all be related to the same site). For MESH the argument should be the path to the OBJ file (the textures contained in the same folder as the OBJ will also be copied)

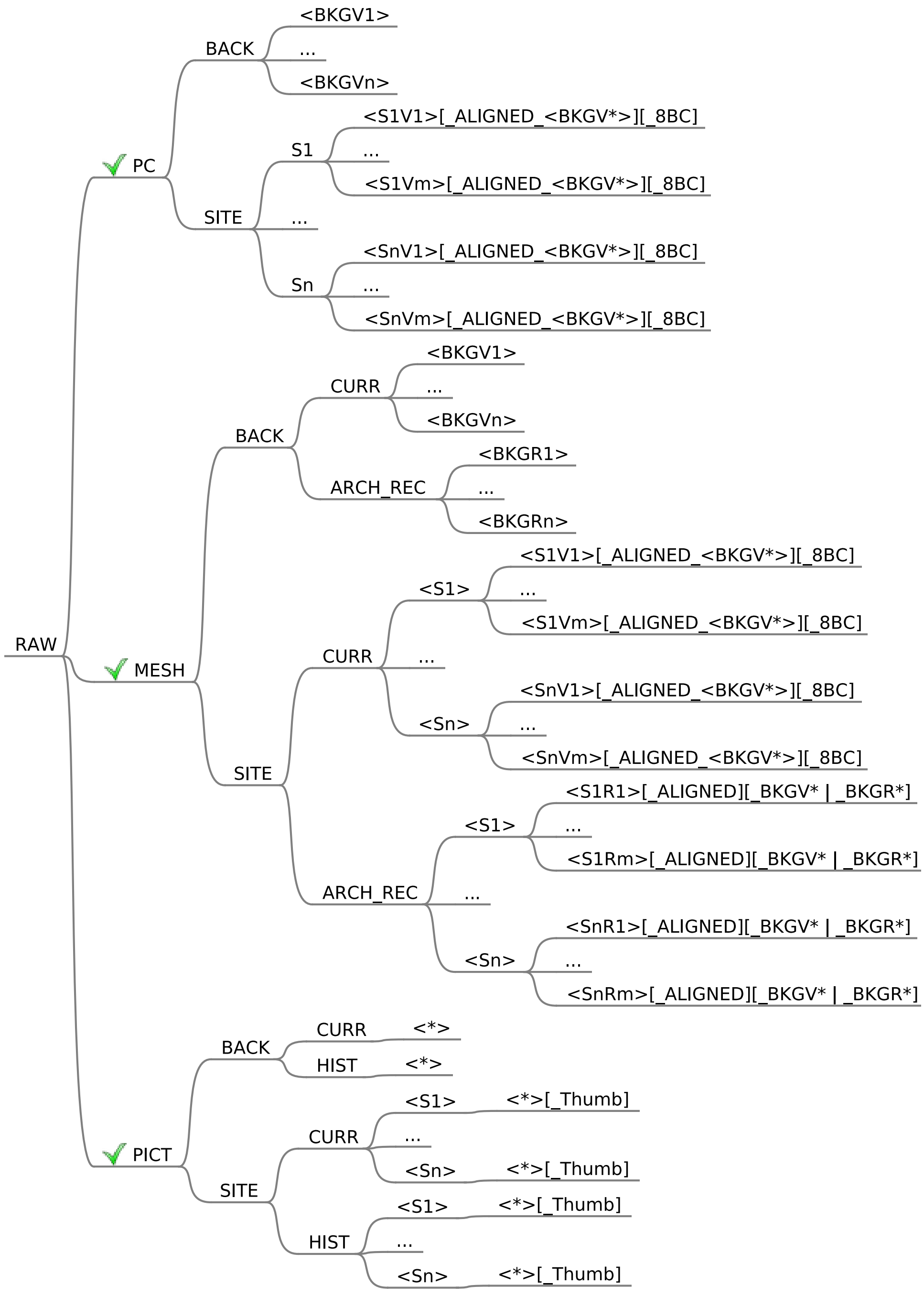

Figure 6: Overview of the data structure: RAW data items

It is important to mention that after adding raw data item one has to update the database in order to properly add the references in the system. See 5.3 to learn more about the database scripts.

```
usage: AddRawDataItem.py [-h] [-i DATA] -k BACK,SITE -t PC,MESH,PICT -f
                         FILE [-p CURR,HIST,ARCH_REC] [-s SRID] [--eight]
                         [-l debug,info,warning,critical,error]
                         [--site SITE]

Add Raw data item to the file structure.

optional arguments:
  -h, --help            show this help message and exit
  -i DATA, --data DATA  RAW data folder [default /home/pattydat/DATA/RAW]
  -s SRID, --srid SRID  spatial reference system SRID [only for MESH SITE]
  --eight               8 bit color [only for PC SITE or MESH]
  -l debug,info,warning,critical,error, --log debug,info,warning,critical,error
                        Log level

required arguments:
  -k BACK,SITE, --kind BACK,SITE
                        Type of item
  -t PC,MESH,PICT, --type PC,MESH,PICT
                        Type of data
  -f FILE, --file FILE  Input file/directory name to copy. For point clouds of
                        sites specify the path to the LAS/LAZ file. For
                        background point clouds specify the folder. For
                        pictures specify the path to the png/jpg/jpeg file
                        (you can also specify a folder with several pictures
                        if they are all related to the same site).
                        For meshes specify the path to the OBJ file (the textures
                        contained in the same folder as the OBJ will also be
                        copied)

required arguments for MESH and PICT:
  -p CURR,HIST,ARCH_REC, --period CURR,HIST,ARCH_REC
                        Period (choose from MESH:CURR,ARCH_REC;
                        PICT:CURR,HIST)

required arguments for SITE:
  --site SITE           Site number
```

4.2.2 Removing raw data items

The *RemoveRawDataItem.py* is used to remove raw data items from the file structure.

```
usage: RemoveRawDataItem.py [-h] -i ITEMID [-d DBNAME] [-u DBUSER] [-p DBPASS]
                            [-t DBHOST] [-r DBPORT]
                            [-l debug,info,warning,critical,error]

Removes a list of Raw data items and their related converted data from the
file structure.

optional arguments:
  -h, --help            show this help message and exit
  -d DBNAME, --dbname DBNAME
                        PostgreSQL DB name viaappiadb]
  -u DBUSER, --dbuser DBUSER
```

```
                        DB user [default $USERNAME]
  -p DBPASS, --dbpass DBPASS
                        DB pass
  -t DBHOST, --dbhost DBHOST
                        DB host
  -r DBPORT, --dbport DBPORT
                        DB port
  -l debug,info,warning,critical,error, --log debug,info,warning,critical,error
                        Log level

required arguments:
  -i ITEMID, --itemid ITEMID
                        Comma-separated list of Raw Data Item Ids Raw data
                        item id (with ? the available raw data items are
                        listed)
```

4.2.3 Listing raw data items

The script *ListRawDataItem.py* lists the raw data items currently in the file structure.

```
usage: ListRawDataItem.py [-h] [-i ITEMID] [-d DBNAME] [-u DBUSER] [-p DBPASS]
                          [-t DBHOST] [-r DBPORT]
                          [-l debug,info,warning,critical,error]

List the Raw data items that are in the DB.

optional arguments:
  -h, --help            show this help message and exit
  -i ITEMID, --itemid ITEMID
                        List the Raw Data Item Ids related to a list of items
                        (comma-separated) [default list all raw data items]
  -d DBNAME, --dbname DBNAME
                        PostgreSQL DB name viaappiadb]
  -u DBUSER, --dbuser DBUSER
                        DB user [default $USERNAME]
  -p DBPASS, --dbpass DBPASS
                        DB pass
  -t DBHOST, --dbhost DBHOST
                        DB host
  -r DBPORT, --dbport DBPORT
                        DB port
  -l debug,info,warning,critical,error, --log debug,info,warning,critical,error
                        Log level
```

4.2.4 Generating OSG data

The script should be run with the *pattydat* user and should be run when some data has changed in the raw data directory. The OSG data is copied automatically in the OSG data tree, with the naming as specified in figure 7.

```
usage: GenerateOSG.py [-h] [-i ITEMID] [-d DBNAME] [-u DBUSER] [-p DBPASS]
                      [-t DBHOST] [-r DBPORT] [-o OSGDIR]
                      [-l debug,info,warning,critical,error]

Generates the OSG data for a raw data item.

optional arguments:
  -h, --help            show this help message and exit
```

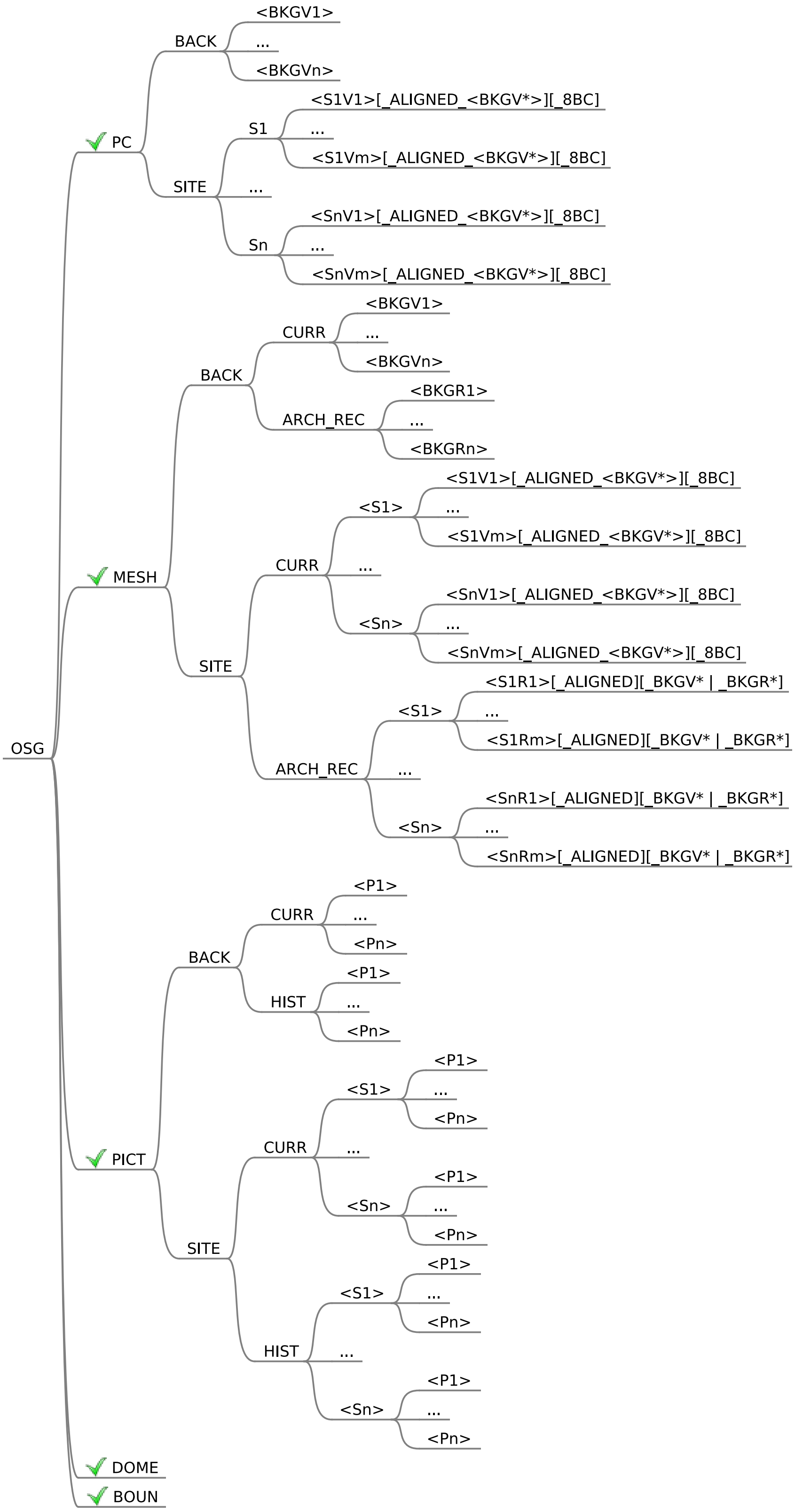

Figure 7: Overview of the data structure: OSG data items

```
  -i ITEMID, --itemid ITEMID
                        Comma-separated list of Raw Data Item Ids [default is
                        to convert all raw data items related to sites that do
                        not have a related OSG data item] (with ? the
                        available raw data items are listed, with ! the list
                        all the raw data items without any related OSG data
                        item)
  -d DBNAME, --dbname DBNAME
                        Postgres DB name [default viaappiadb]
  -u DBUSER, --dbuser DBUSER
                        DB user [default $USERNAME]
  -p DBPASS, --dbpass DBPASS
                        DB pass
  -t DBHOST, --dbhost DBHOST
                        DB host
  -r DBPORT, --dbport DBPORT
                        DB port
  -o OSGDIR, --osgDir OSGDIR
                        OSG data directory [default /home/pattydat/DATA/OSG]
  -l debug,info,warning,critical,error, --log debug,info,warning,critical,error
                        Log level
```

4.2.5 Generating Potree data

The script should be run with the *pattydat* user and should be run when some data has changed in the raw data directory. The Potree data is copied automatically in the Potree data tree, with the naming as specified in figure 8.

```
usage: GeneratePOTree.py [-h] [-i ITEMID] [-d DBNAME] [-u DBUSER] [-p DBPASS]
                         [-t DBHOST] [-r DBPORT] [-o POTREEDIR]
                         [--levels LEVELS]
                         [-l debug,info,warning,critical,error]

Generates the POTree data for a raw data item (ONLY FOR PCs)

optional arguments:
  -h, --help            show this help message and exit
  -i ITEMID, --itemid ITEMID
                        Comma-separated list of PointCloud Raw Data Item Ids
                        [default is to convert all raw data items that do not
                        have a related POtree data item] (with ? the available
                        raw data items are listed, with ! the list all the raw
                        data items without any related POTree data item)
  -d DBNAME, --dbname DBNAME
                        Postgres DB name [default viaappiadb]
  -u DBUSER, --dbuser DBUSER
                        DB user [default $USERNAME]
  -p DBPASS, --dbpass DBPASS
                        DB pass
  -t DBHOST, --dbhost DBHOST
                        DB host
  -r DBPORT, --dbport DBPORT
                        DB port
  -o POTREEDIR, --potreeDir POTREEDIR
                        POTREE data directory [default
                        /home/pattydat/DATA/POTREE]
  --levels LEVELS       Number of levels of the Octree, parameter for
                        PotreeConverter. [default is 4 for Sites and 8 for
```

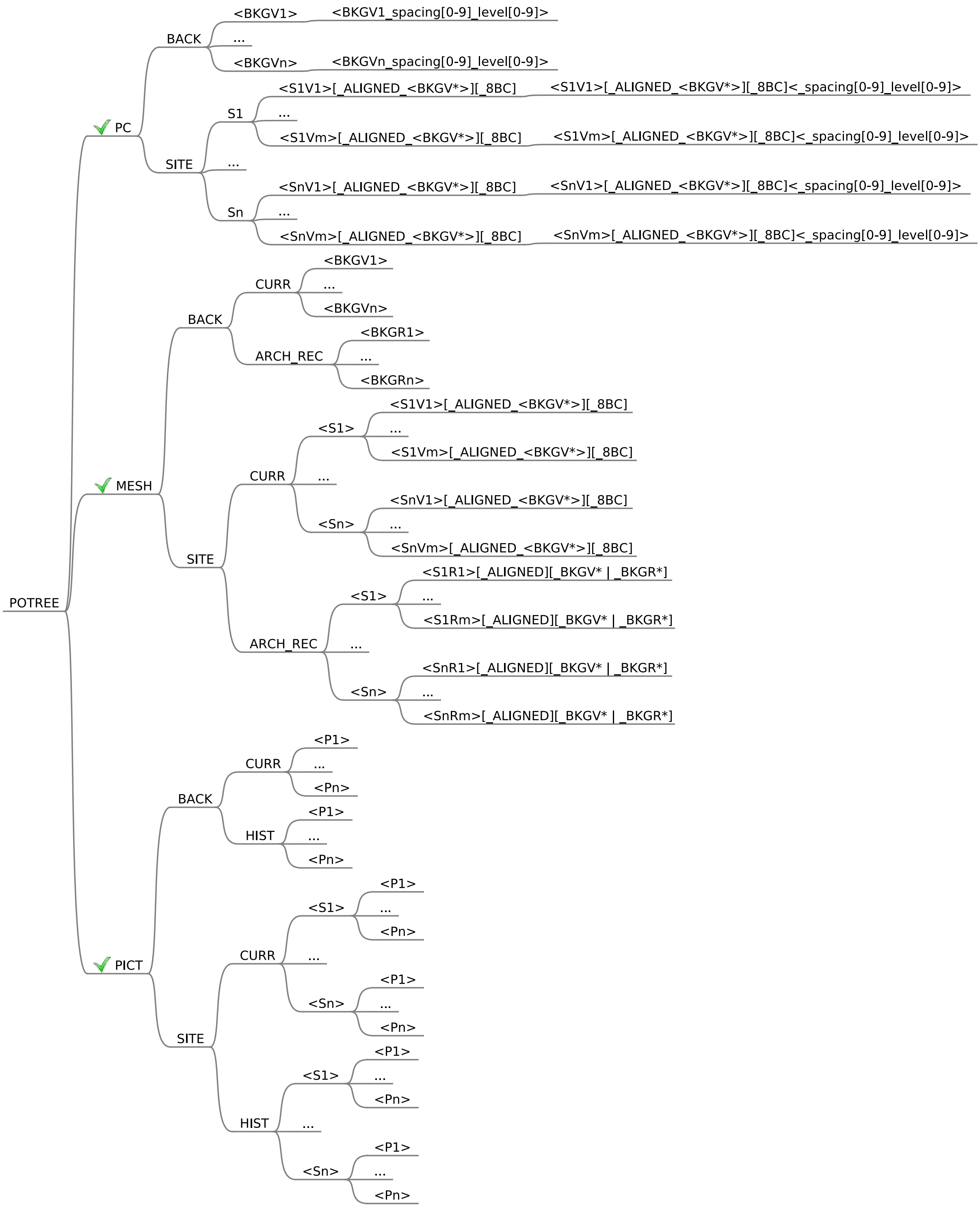

Figure 8: Overview of the data structure: Potree data items

```
                        Backgrounds]
  -l debug,info,warning,critical,error, --log debug,info,warning,critical,error
                        Log level
```

5 Database storage

The *viaappiadb* database is running in the Via Appia Linux server.

The database logical scheme has conceptually two major parts: (a.) data management information and (b.) the attribute data. The data management part has 4 categories: *ITEM*, *RAW*, *POTREE* and *OSG* and the attribute data is represented in the category *ATTRIBUTE*.

Figure 9 contains the Entity-Relationship diagram (ERD) of the *viaappiadb*. In the coming sections each category is described briefly and illustrated. The direct connections between each category are also illustrated.

Note that some of the nodes of the relationships are *0:n* or *0:1* (with black points) instead of the usual *1:n* or *1:1*. This is to illustrate that some sites and objects may have entries in some tables but not in others. For example it is possible to have a site in the item table which has no entry in the attributes table (tbl1_site) because attribute information of that site has not been collected yet.

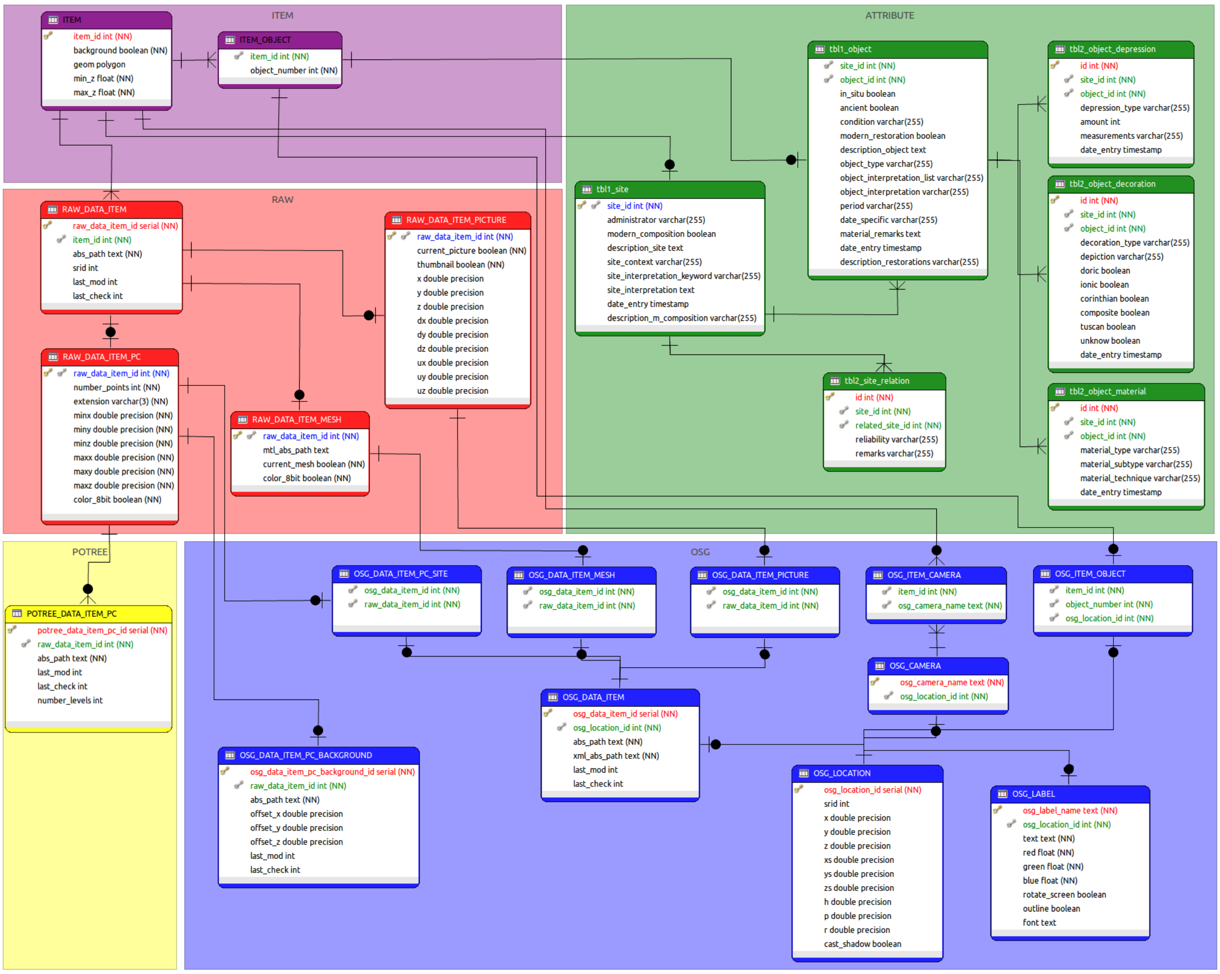

Figure 9: Entity-Relationship diagram of the *viaappiadb* database with its five categories.

5.1 Data management information

This section describes the data management information related categories and for each category we detail its tables.

5.1.1 ITEM

ITEM is a category containing information about the *items* in the studied area. As previously described, an *item* is any entity of interest and it can be either an archaeological site/monument or a background. We use the term background to identify the whole area where the archaeological sites are located. If the item of interest is an archaeological site, in which case the value of the logical field *background* is set to false, the different parts of the item are described as *item_objects* and they are identified by the field *object_number*.

Figure 10 shows the relationship of the *ITEM* category with the other categories. The *ITEM* category is on the top of the categories hierarchy.

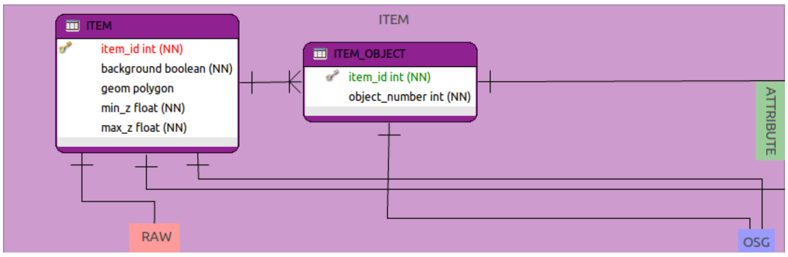

Figure 10: Entity-Relationship diagram of category ITEM and indicated connections with other categories.

5.1.2 RAW

The *RAW* category contains all the references to *raw data* gathered for the items. The most important meta-data is the location in the Data structure (Section 4), stored in the *abs_path* field of the *raw_data_items*. These data can be either point clouds (PC), meshes (MESH) or pictures (PICTURE). Each of these types is represented by a separate table with the specific for that data type properties. The *RAW* category is related to the derived data categories *OSG* and *POTREE* as indicated on Figure 11.

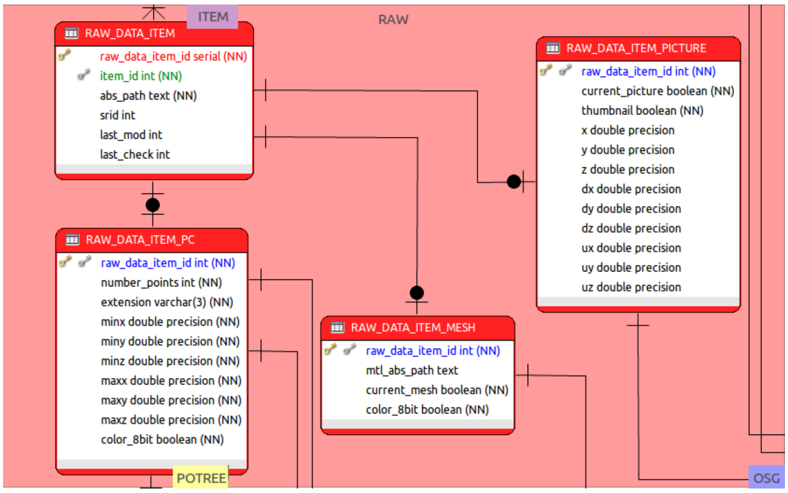

Figure 11: Entity-Relationship diagram of category *RAW* and indicated connections with other categories.

5.1.3 OSG

The *OSG* category contains the references of the OSG *converted data* used for the desktop based viewer. Figure 12 depicts its relationships with the rest of related categories, i.e. the RAW data category (from where it is derived) and the ITEM category. The tables in this category reflect the possible data types: point clouds, meshes and pictures. This category contains meta-data of their data location in the Data Structure (Section 4) as well as information needed for the *visualization*, most importantly the information about the position of the various sites in the background.

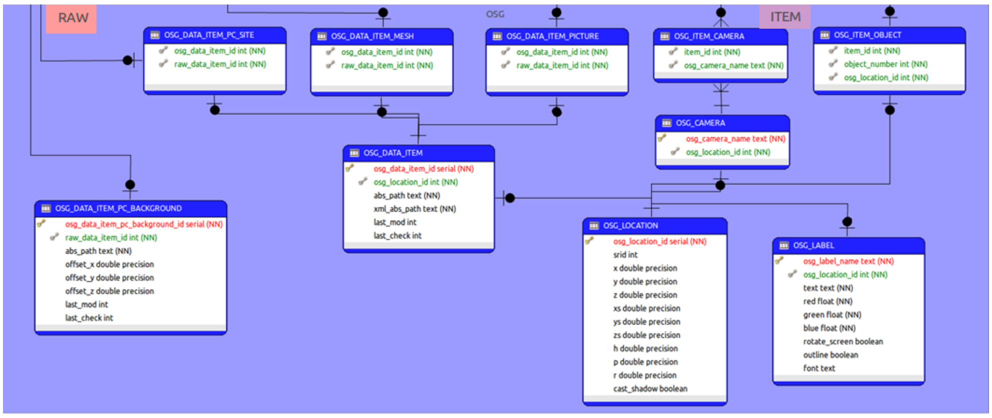

Figure 12: Entity-Relationship diagram of category *OSG* and indicated connections with other categories.

5.1.4 POTREE

The *POTREE* category is illustrated on Figure 13. It is related only to the *RAW* category from which it is derived. Since the POTree web viewer only displays point cloud data only this type of data is considered. In this category we store meta-data information about the location of the data in the Data Structure (Section 4) and the parameters used for the data conversion (*number_levels*).

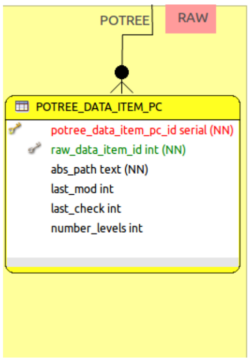

Figure 13: Entity-Relationship diagram of category *POTREE* and indicated connections with other categories.

5.2 Attribute data

The attribute part of the DB is represented only by one category, *ATTRIBUTE* (Figure 14). It is connected only to the *ITEM* category. These are the information collected during the field work and are primarily of research interest to the archaeologists as it contains all domain-related data enabling browsing and filtering of sub-parts of the data of interest.

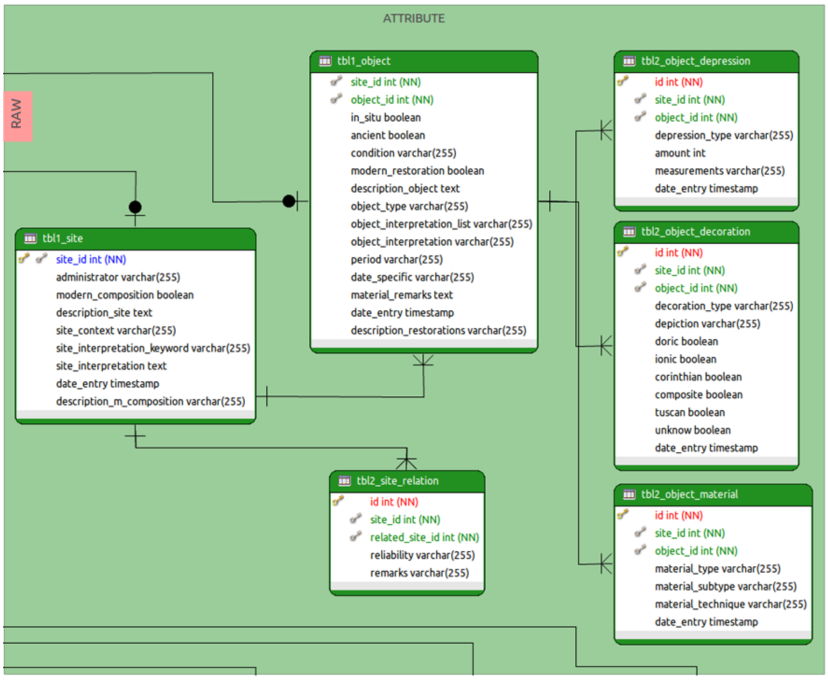

Figure 14: Entity-Relationship diagram of category *ATTRIBUTE* and indicated connections with other categories.

5.3 Database scripts

There is a collection of scripts to manipulate the data in the database. Like for the data storage scripts, it is important to notice that operations modifying the database must be run with *pattydat* user.

5.3.1 Updating the DB

After any change in the directory structure (adding or removing raw data item, generating converted data for OSG or POTree) the user must run the script *UpdateDB.py* in order to let the DB know about the recent changes.

5.3.2 Other database scripts

There are other scripts for updating the footprints, the altitude ranges, the attribute data and to create the database but these are only meant for the administrators and not for regular users so its description is not covered in this manual.

6 Software

As described in section 3 the 4D GIS system has a two tier architecture, the server and the clients. In this section we identify the software required for each of the tiers.

6.1 Server software

The *viaappiadb* database running in the Via Appia Linux server is a PostgreSQL 9.2.8 with PostGIS 2.1.2 and GDAL 1.10.0.

Specific point cloud libraries are also used for the management and processing of point cloud data, concretely LASzip, libLAS and LAStools. In LAStools and libLAS many of their applications share the same name and in these cases the usage of the LAStools ones is preferred. In order to guarantee this we need to set the *PATH* environment variable accordingly, i.e. by setting the LAStools bin folder prior to the libLAS one.

As described in section 4 the data stored in the server needs to be converted to the specific formats required by the two supported visualizations, i.e. the web viewer based on Potree and the Windows desktop viewer/editor based on Open Scene Graph. To perform these conversions we use the *PotreeConverter* (`https://github.com/potree/PotreeConverter`) and the *OSG-Converter* (https://github.com/NLeSC/PattyData/tree/master/OSG/converter) which requires the Open Scene Graph (we currently use 3.2.1). Both converters use Boost (our installed version is 1.55). The data conversion is controlled through python scripts. Hence, it is required python bindings for GDAL, LibLAS and LasZIP.

The web-based viewer requests files from a file server. An NGINX web server is also used to serve files (static content) to the web-based visualization.

6.2 Client software

The web visualization viewer only requires the installation of a modern web browser in the client laptop/desktop (we have used Chrome/Chromium). For the desktop-based visualization the user should follow the instructions at `https://github.com/NLeSC/Via-Appia`.

7 Future work

The foundations for a 4D GIS for Point Cloud data exploration were successfully designed and implemented. In this section we summarize the major future steps to make the system more user efficient, robust and user friendly.

Efficiency

As the number of different objects grows and the number of users increases, it is necessary to have multi-thread scripts for data manipulation, conversion and database operations. For example, the potree converter should be multi-thread. The same holds for database access. Multi-session should be exploited as well as query caching.

On the fly data generation would also open the opportunity to easily integrate new data without having to re-convert all the raw data for a background or site. In the same line, different data organization techniques for Potree, instead the actual octree, should be exploit in case they easy, or boost, on the fly data generation and visualization.

Robustness

At the moment concurrent access for object manipulation is not controlled. For example, in case more than one user changes the object location it leads to data corruption. Hence, as first step it is required a feature which blocks, or discards, other users modifications while the object is being modified. The second step is to allow parallel object manipulation controlled by a version control mechanism.

The addition of new features and the concurrent access by multiple users is a source of robustness issues. The current functionality must remain intact with the addition of new features. Currently the a basic test platform was introduced. However, it is not yet automatized and the set of unit tests is small. In the future all the features, and bug reports, should be covered by unit tests.

User friendly

The data management, such as data upload and data transformation, should be done through a single user interface (UI). The same holds for data visualization. The data conversion and database updates should be triggered by visualization requests on the viewers.

The ultimate goal is an end to end solution where all the parts are glued and automated. Such system should allow an archeologist to be on the field upload a set of photos to the viappia server and wait for an email notification which informs the archeologist about the successful integration of the new data into an existent one. Then with one click the archeologist should be able to visualize/manipulate the new data through the 4D viewer.