newcleo
Futurable Energy

GLOW Current Development State

Davide Manzione[1] davide.manzione@newcleo.com

[1]*new*cleo SrL, Via Giuseppe Galliano 27, 10129 Torino, Italy

August 30th, 2024

Index

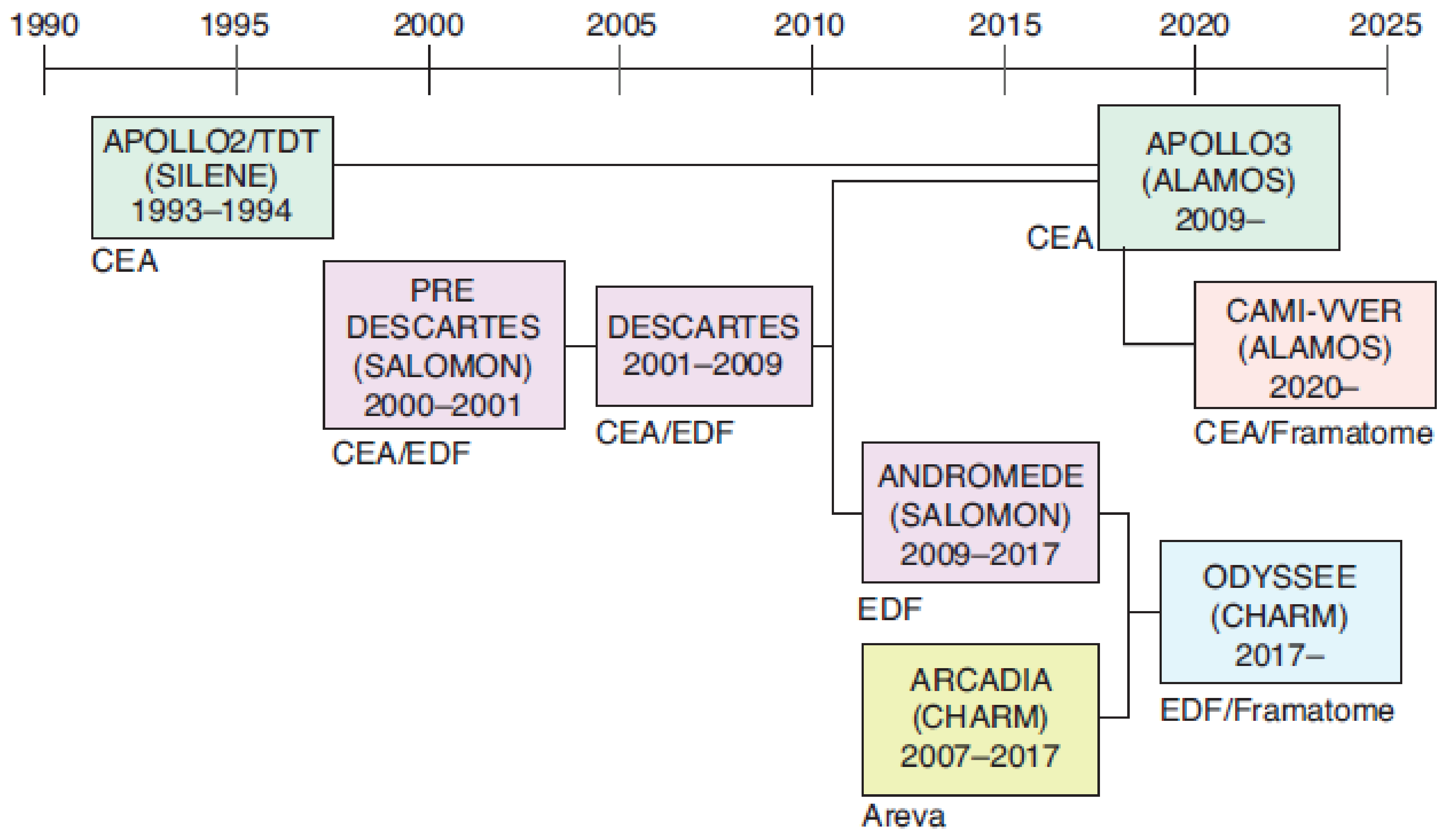
newcleo Futurable Energy

The SALOMON Prototype

The SALOMON Prototype

| R&D on Surface Geometries

- The development of surface geometries for lattice calculations is the result of decades of R&D in France.
- The initial development of surface geometries started in 1993-1994 saw two actors:
  - SILENE – Java GUI for creating non-native surface geometries.
  - TDT – Tool for generating trajectories (tracking) of a surface geometry in the APOLLO2 environment.
- Several other projects followed this (e.g., ALAMOS, SALOMON, etc.).



The SALOMON Prototype

| Characteristics

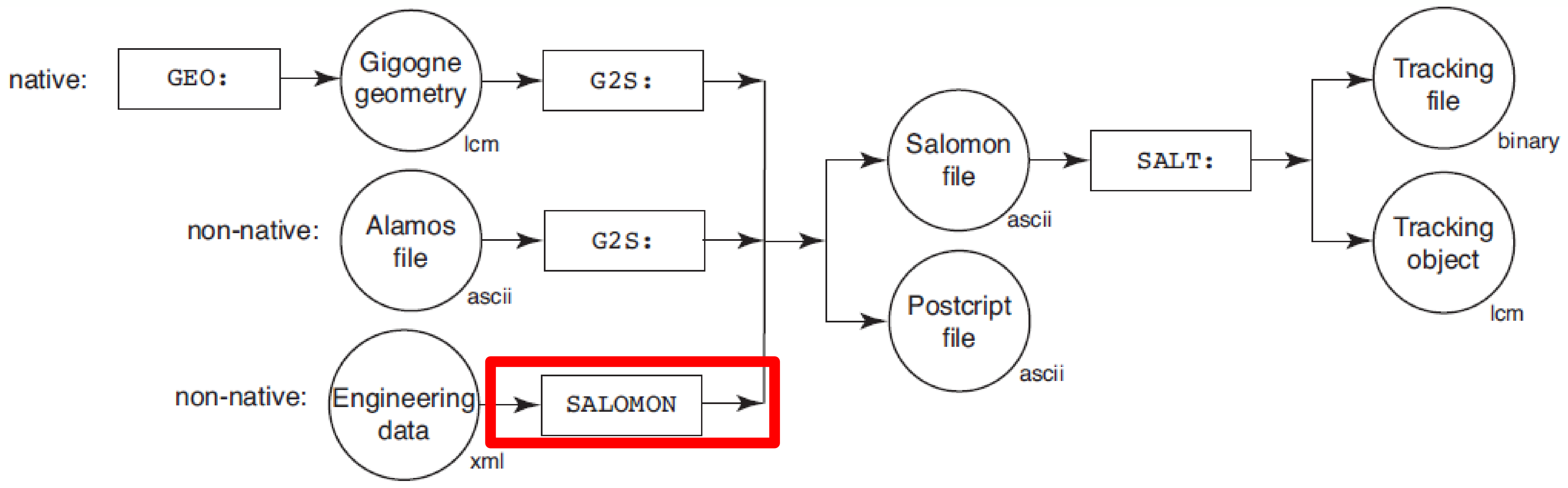
- Part of the DESCARTES project, whose aim was to implement a new assembly calculation scheme by means of rapid prototypes based on open-source components (SALOME, Python2, DRAGON2).
- **SALOMON main features**:
  - tool for creating non-native surface geometries for the TDT solver (trajectories tracking);
  - based on the API of the GEOM module of the SALOME platform;
  - assembled by means of about 1900 Python2 lines.

The SALOMON Prototype

| Workflow

- The SALOMON application proceeds in two steps:

1. Given an input XML file describing the lattice geometry in terms of cells and materials associated with each one, a surface representation is generated automatically using the APIs of SALOME's GEOM module.
2. Translation of the GEOM lattice representation into a file in the TDT format used by the APOLLO2 (MOC) or DRAGON5 (SALT module) codes.



The SALOMON Prototype

| Input XML File

- It provides a description of the lattice layout (see <*ComponentIDList*>) in terms of its cells.
- Each cell (see <*Cell*>) type has an ID and is described by means of:
  - a list of materials (see <*MaterialList*>);
  - an ID (see <*GeomCellID*>) associated to an object (see <*GeomCell*>) in the XML structure describing the cell geometry (cell width/height, radii of the cell inner circles).

```
<Header ID="START">
<HeaderComponentID> MyLattice_HTP900_assemblage_mox__CellConfig__STD_ </HeaderComponentID
<BoundCond> mirror </BoundCond>
<CalcSym> 4 </CalcSym>

<Compound ID="MyLattice_HTP900_assemblage_mox__CellConfig__STD_">
<MainComponentID> MyLattice_HTP900_assemblage_mox__CellConfig__STD___PLT </MainComponentI
<XCoordinateList> 0.00000000e+00 </XCoordinateList>
<YCoordinateList> 0.00000000e+00 </YCoordinateList>
</Compound>

<Lattice ID="MyLattice_HTP900_assemblage_mox__CellConfig__STD___PLT">
<XNbComponents> 3 </XNbComponents><YNbComponents> 3 </YNbComponents>
<ISym> 1 </ISym>
<ComponentIDList>
   Lame_C  LameH  Lame_C
   Lame_V  C0101  Lame_V
   Lame_C  LameH  Lame_C
</ComponentIDList>
</Lattice>
```

```
<Cell ID="Lame_C">
<MaterialList> TSTR.'TMil_MOC'.'MODE' </MaterialList>
<GeomCellID> GLame_Coin </GeomCellID>
</Cell>

<Cell ID="Lame_V">
<MaterialList> TSTR.'TMil_MOC'.'MODE' </MaterialList>
<GeomCellID> GLame_Verticale </GeomCellID>
</Cell>

<Cell ID="LameH">
<MaterialList> TSTR.'TMil_MOC'.'MODE' </Materiall
<GeomCellID> GLame_Horizontale </GeomCellID>
</Cell>
```

```
<GeomCell ID="GLame_Coin">
<Type> uniform </Type>
<XSize> 0.042 </XSize>
<YSize> 0.042 </YSize>
<XNbMesh> 1 </XNbMesh>
<YNbMesh> 1 </YNbMesh>
</GeomCell>
```

```
<GeomCell ID="MAILCOMB">
<Type> standard </Type>
<XSize> 1.26000000e+00 </XSize>
<YSize> 1.26000000e+00 </YSize>
<OuterRadiusList> 0.288712 0.365195 0.397962 0.4083 0.4165 0.4775  </OuterRadiusList>
<NbSectorList> 1 1 1 1 1 1 </NbSectorList>
<InitialAngleList> 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 </Init
</GeomCell>
```

newcleo Futurable Energy

The SALOMON Prototype

| Execution

- SALOMON is executed by running a specific shell script (*./SALOMON.sh*) with the following arguments:
    - -o <OutputFolder>, used to specify where the output files should be produced;
    - <InputXMLFile>, the file with the lattice information to process.

```
./SALOMON -o Outputs/ InputXmlData/UOX_TBH_1cell.xml
```

- The shell script builds a temporary Python script whose objective is to run the main file of SALOMON by passing the command line arguments.
- This temporary Python script is passed as argument to SALOME, so to perform the surface geometry conversion.
- It also handles the two main SALOMON execution modes:
    - **GUI mode** – It opens the SALOME GUI so that the lattice geometrical representation can be inspected directly;
    - **Batch mode** – SALOME is run without the GUI and SALOMON is executed to produce the output files.

The SALOMON Prototype

| Output Files

- After extracting the geometry information and performing the analysis on the lattice, SALOMON produces two output files:
  - *.dat* file, describing the lattice surface geometry; it can be used by DRAGON5 directly;
  - *.mat* file, providing a correspondence between the name of the materials and the indices used in the *.dat* file.

The SALOMON Prototype

| Criticalities

Geometries — Support only to **cartesian cell and lattice geometries**:

- No hexagonal cases can be handled.
- No support for specific lattice symmetries of interest (S30, RA60, R120 and COMPLETE).

Dependencies — Based on **outdated versions** of open-source tools:

- SALOME v6.6.0
- Python2

Usability — **Different configuration steps** are needed:

- Installation of SALOME and all its pre-requisites.
- Necessity to run a shell script for setting SALOMON environment variables.
- Necessity to include additional environment variables in the *.profile* file.

The GLOW Project

The GLOW Project

| Development Requirements

- The GLOW (Geometry Layout for OpenCascade Workshops) generator is a tool currently being jointly developed by the Polytechnique of Montréal and *new*cleo.
- Its aim is to offer an open-source alternative to ALAMOS for defining non-native geometries for DRAGON5.
- The main **development requirements** of GLOW are:
  - The output file (*.dat*) providing the surface geometry representation shall be in the TDT APOLLO2 format.
  - GLOW shall be based on Open Cascade, rather than on SALOME.
  - The same SALOMON’s two-stage approach, as proposed by Yann Pora, shall be used.
  - The first production version shall target hexagonal geometries.

The GLOW Project

| Development Steps

- Four steps have been identified to drive the development:

1. Analysis of the **SALOMON prototype workflow**.
2. **Adaptation** of the SALOMON prototype **to Open Cascade functions**, instead of the SALOME ones (still considering cartesian geometries).
3. Creation of an XML syntax to **describe assemblies** and colorsets **for hexagonal geometries**.
4. **Generalization** of the prototype to address to all kind of hexagonal geometries (S30, RA60, R120 and COMPLETE) with 2D housing and/or stiffeners.

The GLOW Project

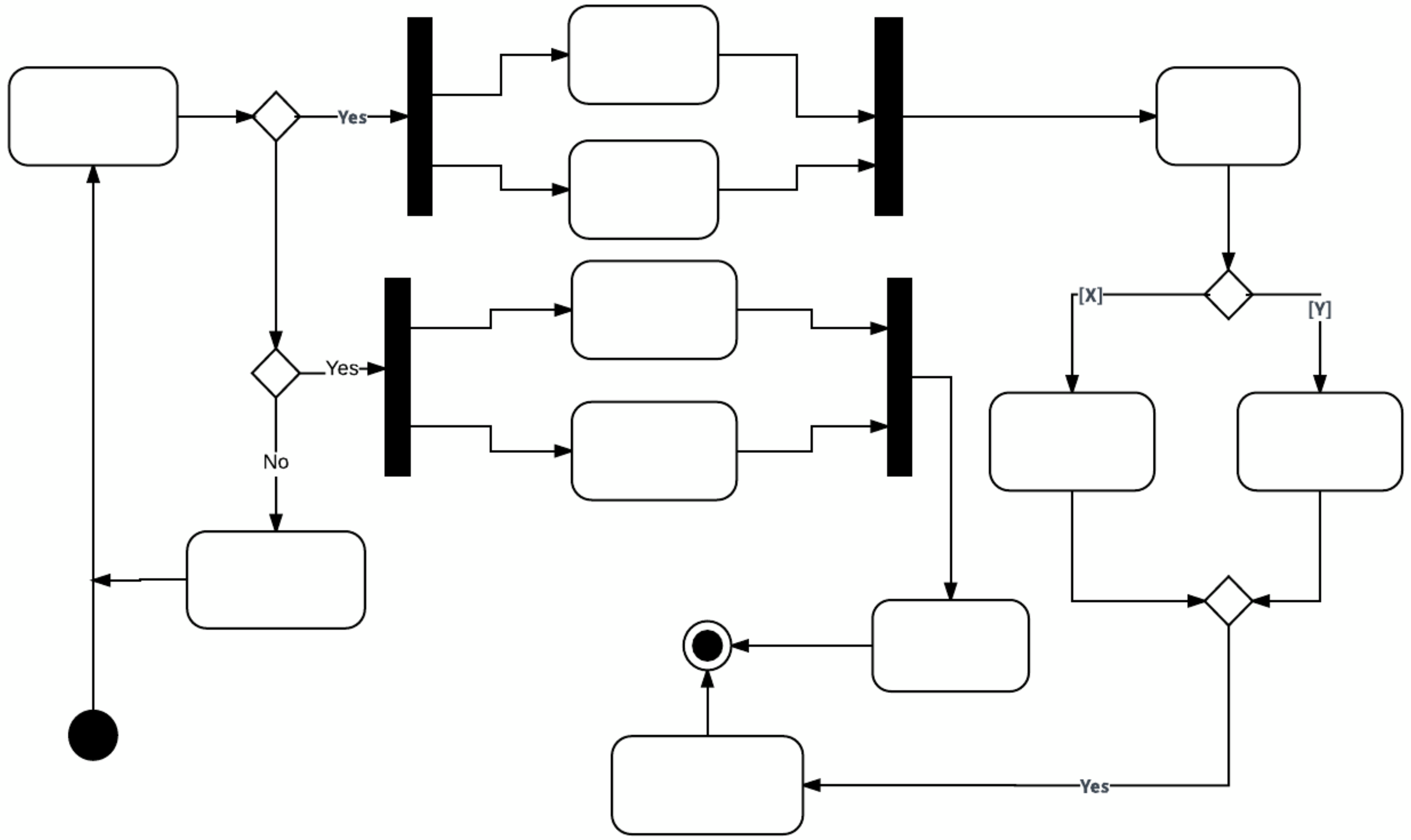
| 1. SALOMON Workflow Analysis

- To get a clear understanding of the SALOMON workflow, the original version has been ported to comply with the latest versions of both SALOME (v9.12.0) and Python3 (v3.11).
- Some additional activities have been performed to support a clean and functioning version of SALOMON:
  - Bug fixing.
  - Moved project to **Git local repository**.
  - Added Python docstrings to **document the code in English**.
  - **SALOMON.sh script restructure** to include:
    - All the paths to the needed SALOME modules (setting of environment variables are no longer required).
    - Proper handling of the two SALOME modes according to what required by the latest version.
- All the modifications and the updates introduced to the old SALOMON version has been tested so that the output geometry conversion still produces compatible results for the MOC analysis.

The GLOW Project

| 1. SALOMON Workflow Analysis Representation

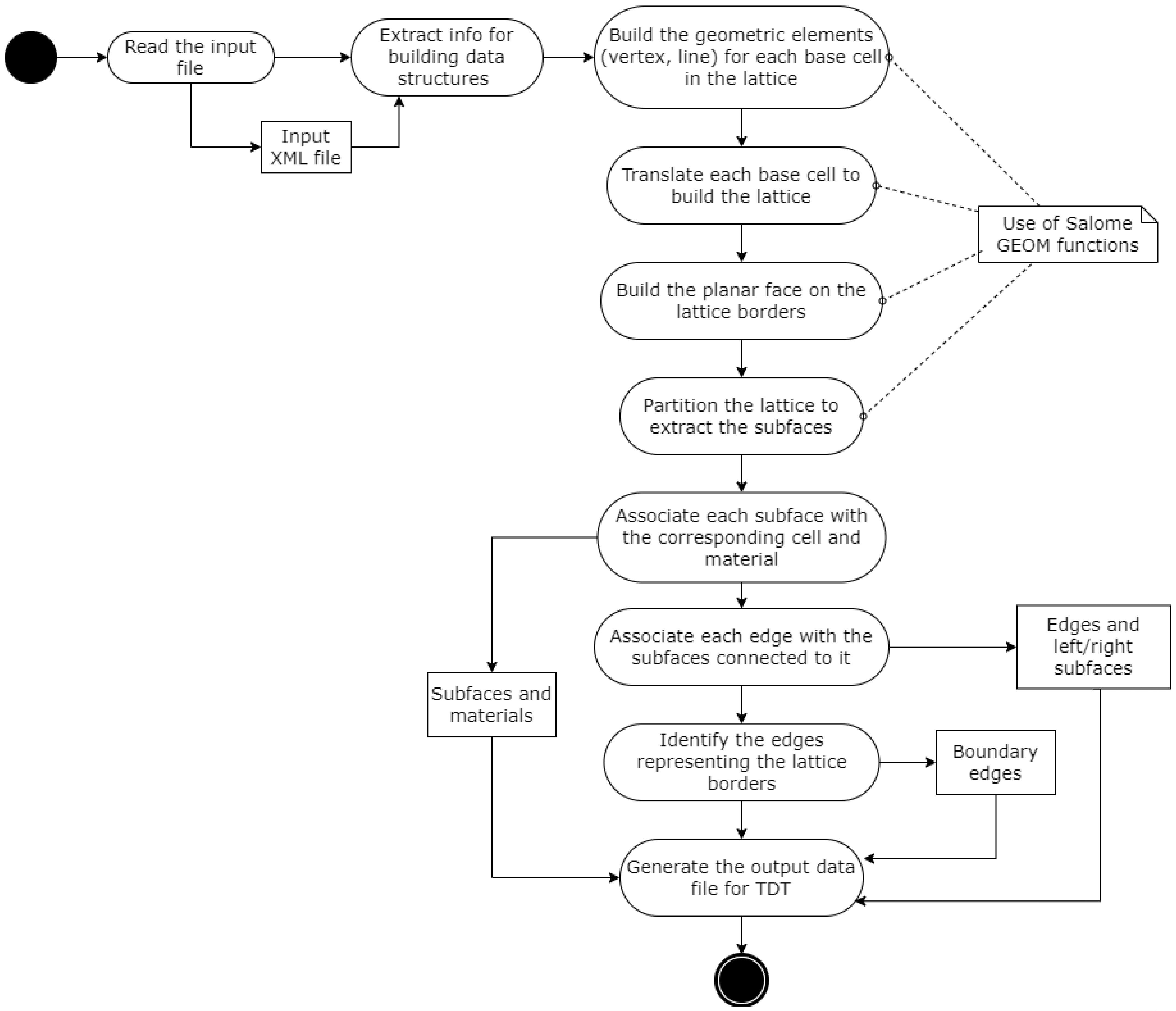
- Having updated and made SALOMON work with the latest dependencies, a proper analysis of its workflow has been realised.
- Use of **UML** (Unified Modelling Language) methodologies to address the task:
  - It provides a **standardized way** for describing software in a **visual** format.
  - Different set of diagrams are present to help software developers in documenting the elements and features of a software.
- **Activity diagrams** have been derived for this purpose. They show the **sequence of** performed **operations** in a block format, with arrows connecting two or more operations logically.



The GLOW Project

| 1. SALOMON Workflow Analysis Representation

- Different level of decomposition and complexity can be represented.
- The aim is to understand **how the software operates** so to:
  - **identify** any **criticality** to address to by restructuring the code;
  - **identify** all the **geometrical operations** performed by the GEOM module of SALOME to replicate them using the Open Cascade functionalities.



newcleo Futurable Energy

The GLOW Project

| 2. Adaptation to Open Cascade

- The Open CASCADE Technology (OCCT) is an **object-oriented C++ class library** designed for domain-specific CAD applications.
- It provides functionalities to address **2D/3D geometric modelling**.
- It represents the **base** upon which the SALOME functions are built.
- To use OCCT functions within GLOW, a proper **Python wrapper library is needed**: the *pythonocc* library has been selected.
- *pythonocc* provides a **full access from Python to** almost all the **Open Cascade C++ classes**. Classes and methods/functions share the same names, and the same signature. In addition, it comes with 3D visualization for the most famous Python GUI libraries.
- Having identified all the used SALOME functions, they have been substituted with the corresponding OCCT ones from the *pythonocc* library.
- **N.B.** A **test phase is still required** so to assure that the output geometry conversion produces compatible results for the MOC analysis.

The GLOW Project

| 2. SALOME VS Open Cascade

SALOME

- **Pros:**
  - It comes with a utility that checks the correct installation of all prerequisites.
  - Functions can be used directly from Python
  - It comes with a GUI for both visualizing and building of 2D lattice geometries
  - Presence of an online comprehensive documentation.
  - Easiness to use within the code.
- **Cons:**
  - Users need to handle the prerequisites installation by themselves
  - Lack of information on type of objects from the code.

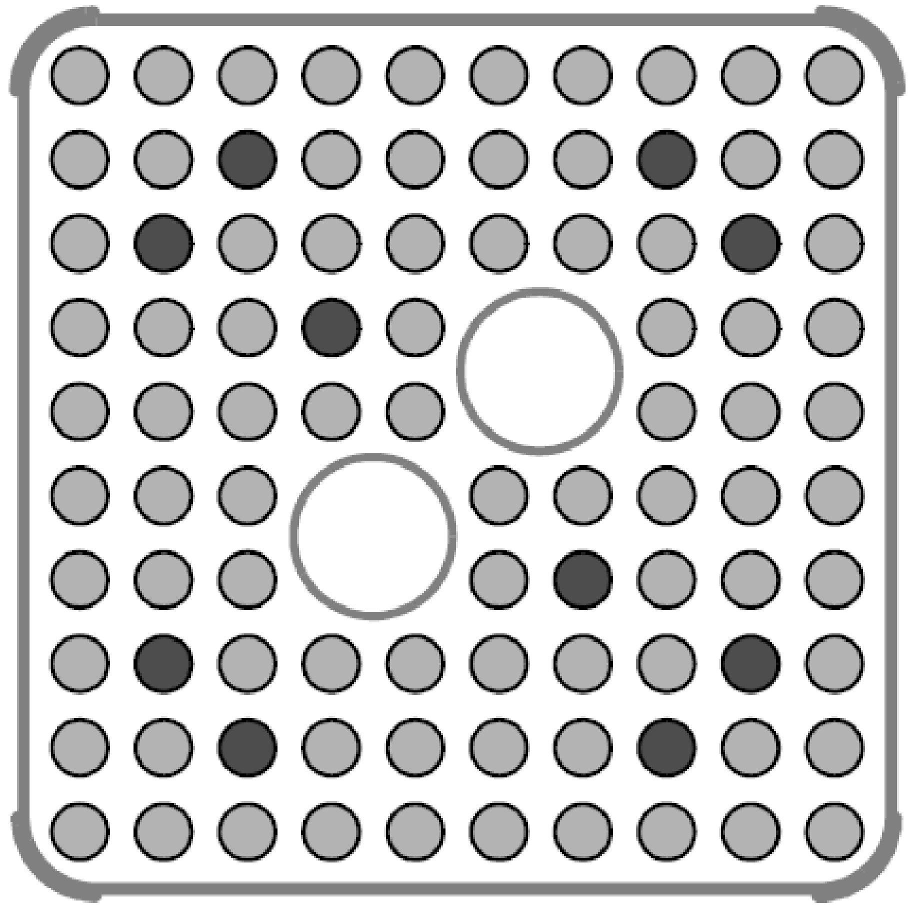
Open Cascade

- **Pros:**
  - It comes with a GUI for visualizing the built 2D lattice geometries
  - Presence of an online comprehensive documentation.
  - Type of objects are well documented in the Python wrapper code.
- **Cons:**
  - Users need to handle the installation procedure of both Open Cascade and *pythonocc*.
  - The *pythonocc* wrapper of OCCT functions is needed.
  - Presence of several libraries and functions for building the same geometrical objects.

The GLOW Project

| Discussion Points

- XML file describing the lattice geometry can **cover only specific cases**.
- **Complex lattice geometries** made of cells with different dimensions or built by means of Boolean operations (fuse, intersection, cut) **cannot be handled**.
- No possibility to **draw** the geometry **and convert** it directly.
- Decision about the use of SALOME or Open Cascade needs to be taken.



Conclusions

5. Conclusions

Key Takeaways

- The GLOW project aims at being an open-source alterative to ALAMOS for converting a geometry lattice representation in the TDT format for successive analyses in DRAGON5.
- Up to now, it is based on the SALOMON prototype two-step approach, i.e. extracting data from an XML to build the geometry elements and process them to produce a file for DRAGON5.
- The development has been subdivided into 4 steps, the first two of them being already addressed.
- Criticalities have currently emerged, and decisions need to be taken on the direction to take.

Thank you for your attention