02 | 07 | 2023

Basic of Apothesis

Apothesis is designed to be a generalized, psudo-3D kinetic Monte Carlo (kMC) software for deposition processes based on the solid-on-solid approximation. Apothesis is developed in C++.

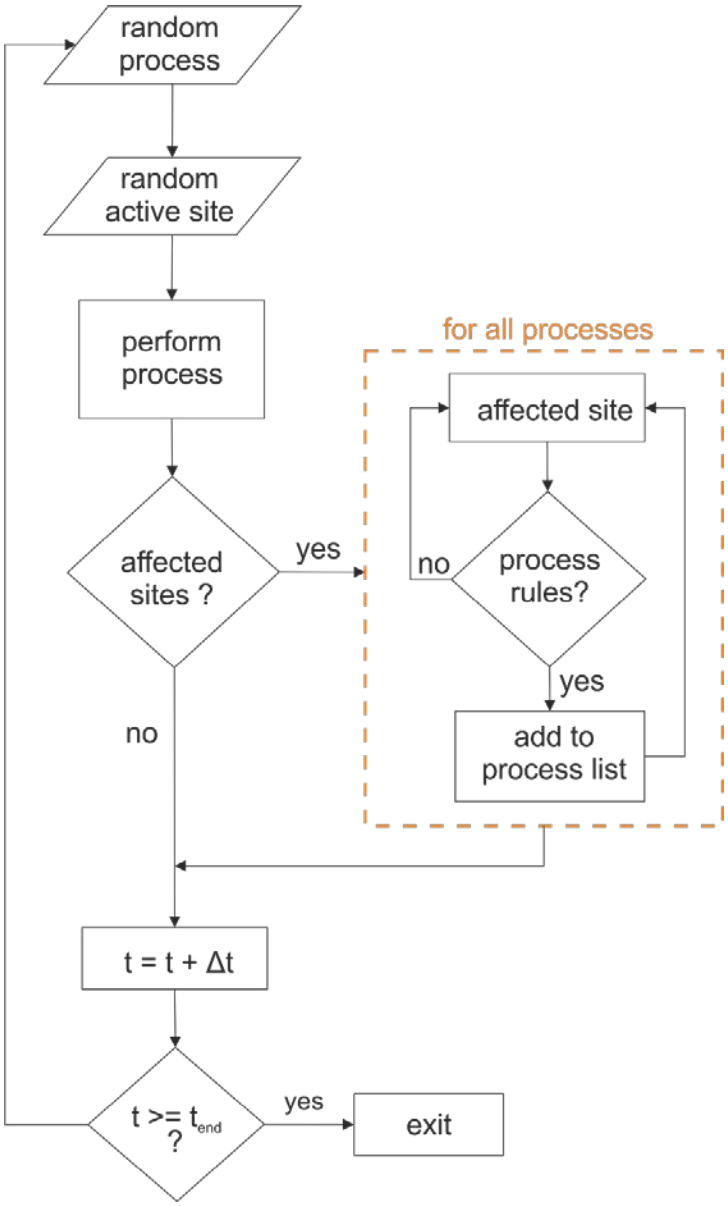

Figure 1 The computational "backbone" of Apothesis software. The algorithm starts by picking a random process and a random active site of this process. The process is performed and if it affects other sites a loop over all process is

performed in order to update their lists according to their “rules” until all process lists are updated. After that the time is updated until the end time, t_{end} is reached.

The main components of Apothesis are the lattice and the processes. The lattice is a predefined number of positions, termed as sites, where the different processes can perform on. Depending on the material it can be simple cubic, Body Cubic Centered (BCC), Face Cubic Centered (FCC) etc. A process is an act on a site of the lattice. If a process can perform on a site, this site is termed as “activated” otherwise it is “blocked”. An Apothesis process emulates a physical/chemical process e.g. the adoption of a species on an active site.

The sites of the lattice where a process can perform are kept on a list. The size of this list multiplied by the rate constant of the process is the process rate. In the course of a kMC simulation, each process list must be updated in order for the rates to be computed correctly. This make this part of the code crucial in terms of the validity of the software and in term of speed, since this is the most computational demanding part of kMC method in general. In order to generalize Apothesis, this part of the software should be very carefully designed and developed.

We tackled this issue by designing an abstract **process** class which contains three primary functions, the “init”, the “rules” and the “perform” functions. The “init” function is the one called for the initialization of the process and is responsible to match a function pointer to a specific function for the “rules” and “perform”. The “rules” function defines, computationally, when a process can be performed or not in a site and the “perform” is the process act on the lattice site. We take as an example the desorption of a particle of 4 nearest neighbors. The “rules” in this case would be to count the nearest neighbors of the site and if it is 4 then add it in its list. The “perform” would be to remove one particle (decrease site’s height) and mark the sites that are affected which in this case are all 4 nearest neighbors. A loop is the performed to check, through the “rules”, if the affected sites must be removed or added to any of the other process lists. In that way, the screening is performed in only a small number of sites, and not the entire lattice, reducing significantly the computational time. In this way, we construct a generalized framework where every process can be described independently of the lattice and thus, producing a robust computational environment for kMC computations. The computational “backbone” of Apothesis can be seen schematically in Fig. 1.

How to add a new process in Apothesis

Four basic processes are supported in Apothesis:

1. Adsorption
2. Desorption
3. Diffusion

4. Surface reactions

Each of these processes are implemented in classes located under the **processes** directory.

For the shake of simplicity we will describe adsorption but the same holds for all the other processes.

Adsorption

Adsorption is one of the most basic processes since it describes the sticking of an atom from the gas phase to the surface (lattice). Adsorption is implemented in the files **adsorption.h/cpp** under processes directory.

In the header we have these three basic functions:

```
void init( vector<string> params ) override;
bool rules( Site* ) override;
void perform( Site* ) override;
```

Lets see every one of them separately:

The rules function

```
bool Adsorption::rules( Site* s )
{
   (this->*m_fRules)(s);
}
```

The perform function

```
void Adsorption::perform( Site* s )
{
   (this->*m_fPerform)(s);
```

```
}
```

Both functions are simple calls to a function pointer.

The init function based on the input for this process maps the function pointers to the relative functions. For example, if the user uses in the input file the line:

A + * -> A* constant 0.5

this means the adsorption of A on a site of the lattice with a constant rate constant. According to this in the init function the yellow highlight part of the function will be called which points to a constant adsorption type. The m_fType points to a function for calculating the rate constant (e.g. simple, constant, Arrhenius etc.)

Then the rules must be set. Depending if the atom is part of the growing surface or not and if the atom needs more than 1 sites to adsorb (currently only 2 are supported) the rules are set.

Finally, as in the case of rules the perform is selected.

The init function

```
void Adsorption::init( vector<string> params )
{
 m_vParams = params;

  //Check the type of this adsorption type.
  //Nore: The first argument must always be the type.
  m_sType = any_cast<string>(m_vParams[ 0 ]);
  if ( m_sType.compare("simple") == 0 ){
     m_dStick = stod(m_vParams[ 1 ]);
     m_dF = stod(m_vParams[ 2 ]);
     m_dCtot = stod(m_vParams[ 3 ]);
     m_dMW = stod(m_vParams[ 4 ]);
```

```
    m_fType = &Adsorption::simpleType;
}
else if (  m_sType.compare("constant") == 0  ) {
    m_dAdsorptionRate = stod(m_vParams[ 1 ]);

    m_fType = &Adsorption::constantType;
}
else {
    m_error->error_simple_msg("Not supported type of process: " + m_sType );
    EXIT
}

//Create the rule for the adsoprtion process.
if ( m_iNumSites == 1 && isPartOfGrowth( m_sAdsorbed ) ){
    setUncoAccepted( true );
    m_fRules = &Adsorption::uncoRule;
}
else if ( m_iNumSites > 1 && isPartOfGrowth( m_sAdsorbed ) )
    m_fRules = &Adsorption::basicRule;
else if ( m_iNumSites == 1 && !isPartOfGrowth( m_sAdsorbed ) )
    m_fRules = &Adsorption::multiSpeciesSimpleRule;
else if ( m_iNumSites > 1 && !isPartOfGrowth( m_sAdsorbed ) )
    m_fRules = &Adsorption::multiSpeciesRule;
else {
    m_error->error_simple_msg("The rule for this process has not been defined.");
    EXIT
}

//Check what process should be performed.
//Adsorption in PVD will lead to increasing the height of the site
```

```
  //Adsorption in CVD/ALD will only change the label of the site
  if ( m_iNumSites == 1  && isPartOfGrowth(m_sAdsorbed) )
     m_fPerform = &Adsorption::signleSpeciesSimpleAdsorption;
  else if ( m_iNumSites > 1  && isPartOfGrowth( m_sAdsorbed ) )
     m_fPerform = &Adsorption::signleSpeciesAdsorption;
  else if ( m_iNumSites == 1 && !isPartOfGrowth(m_sAdsorbed) )
     m_fPerform = &Adsorption::multiSpeciesSimpleAdsorption;
  else if ( m_iNumSites > 1 && !isPartOfGrowth(m_sAdsorbed) )
     m_fPerform = &Adsorption::multiSpeciesAdsorption;
  else {
     m_error->error_simple_msg("The process is not defined | " + m_sProcName );
     EXIT
  }

  (this->*m_fType)();
}
```

The constatType function

```
void Adsorption::constantType(){
  m_dProb = m_dAdsorptionRate*m_iNumVacant;
}
```

sThus, in order for the user to set up a new process, she/he must change the init function and create the rules and perform function for this process.