Cyclus, an agent-based fuel cycle simulator

Brief Overview

Jin Whan Bae, Kathryn Huff

University of Illinois at Urbana-Champaign

May 21, 2018

ILLINOIS

Agent-based Framework

- Cyclus is agent-based, which means it's very modular
- User can develop / plug in facilities
  - User can 'design' their own fuel cycle
  - Highly customizable

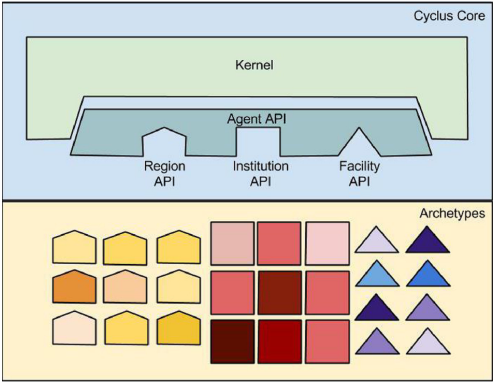
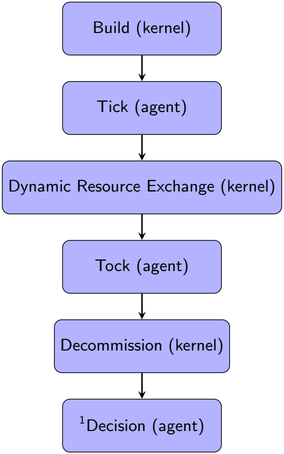


Figure 1: Modular Design of Cyclus

Timestep Execution

A simplified explanation: Each timestep:



[1] Decision phase is planned to be added in the next release.

General Info

- Written in: C++, Python
- Input file: xml, json, python
- Output file: .sqlite, .hdf5

Terminology

- **Archetypes**: A collection of logic and behavior which can be configured into a prototype which can then be instantiated in simulation as a agent. Archetypes are represented as C++ classes that inherit from the base cyclus::Agent class. (e.g. Reactor module, Sink module)
- **Prototypes**: Archetype + parameters (e.g. Reactor with input-defined `name, cycle time, assembly size, core size etc`)
- **Agents**: Every single 'entity' in play during simulation (Region, Institution, Facility)

Terminology

- **Region**: The group agent that is a collection of institutions (Can manage / control regions)
- **Institution**: Agent that manages facilities (Can deploy, decommission facilities)
- **Facility**: The agent that 'trades' and does calculations (Trades material and transmutes, separates)

Extensions - Archetypes

Since Cyclus is an extensible framework, anyone can develop a new archetype and plug-and-play. (Institution, region, facility otherwise.)

- Cycamore: Sink, Storage, Recipe Reactor, Fuelfab, Enrichment, Source, DeployInst, Mixer, Separations, GrowthRegion
- *D3ploy: Demand-driven deployment Institution (NEUP 16-10512)
- *CYBORG: Reactor depletion analysis tool using ORIGEN
- *CYDER: A CYclus Disposal Environment and Repository object.
- *CORRM: Continuous On-line Reprocessing Reactor Module.
- *Pyre: Pyroprocessing module with non-proliferation metrics
- *Peddler: Simulate trucks and transport material between facilities.
- And more..

* Third party module in active development.

Extensions - Analysis / Drivers

There are other tools to help visualization / output data analysis of Cyclus.

- RICKSHAW: Automated stochastic driver for Cyclus
- Cymetric: Extracts important fuel cycle metrics
- Analysis: Collection of functions to extract metrics (e.g. natU usage, trade between two facilities, etc.)
- Cycmap: GIS visualization tool for Cyclus
- Cyclist: GUI for Cyclus (DEPRACATED)

Installation - Binary

Better, more thorough explanations are in `fuelcycle.org`

- Windows: N/A
- MacOS: `conda install -c conda-forge cyclus cycamore`
- Linux: `conda install cyclus cycamore`

Installation - Build from Source

All source files are open-source, and available on Github.
`github.com/cyclus/cyclus` and `github.com/cycamore/cycamore` has the source files, and guides

1. Clone repository (`git clone [url]`)
2. Install dependency (see github guide README)
3. `python install.py`

Installation - TroubleShooting

Look for your error message or make a new post in the following Cyclus communities:

1. Github Issue in `github.com/cyclus/cyclus`
2. Cyclus google user group
3. Email jbae11@illinois.edu (me)

Structure

1. `Control`: Simulation Definition
2. `Archetypes`: List of available archetypes
3. `Facility`: Facility prototypes - define parameters of archetypes
4. `Region`: Region agents
5. `Institution`: Institution agents (inside Region definition)
6. `Recipe`: recipe definitions

Control

```xml
<control>
  <duration>2280</duration>
  <startmonth>1</startmonth>
  <startyear>1970</startyear>
  <decay>manual</decay>
</control>
```

Archetypes

```
<archetypes>
  <spec>
    <lib>cycamore</lib>
    <name>Source</name>
  </spec>
  <spec>
    <lib>cycamore</lib>
    <name>Sink</name>
  </spec>
  <spec>
      <lib>cycamore</lib>
      <name>Reactor</name>
  </spec>
  <spec>
    <lib>agents</lib>
    <name>NullRegion</name>
  </spec>
  <spec>
    <lib>agents</lib>
    <name>NullInst</name>
  </spec>
  <spec>
    <lib>cycamore</lib>
    <name>DeployInst</name>
  </spec>
  <spec>
    <lib>cycamore</lib>
    <name>Separations</name>
  </spec>
</archetypes>
```

Facility - Cycamore::Separations

```xml
<!-- La Hague Model from Schneider & Marignac -->
<!-- Since 1976 -->
  <facility>
    <name>LA_HAGUE</name>
    <config>
      <Separations>
         <feed_commods>   <val>cooled_french_uox_waste</val> </feed_commods>
         <feed_commod_prefs> <val>20.0</val> </feed_commod_prefs>
         <feedbuf_size>91600</feedbuf_size>
         <!-- 1100 tons/year 91.6 tons/timestep -->
         <throughput>91600</throughput>
         <leftover_commod>lahague_raffinate</leftover_commod>
         <leftoverbuf_size>91600</leftoverbuf_size>
         <streams>
          <item>
            <commod>french_uox_Pu</commod>
            <info>
              <buf_size>91600</buf_size>
              <efficiencies>
                <item>
                  <comp>Pu</comp> <eff>.998</eff>
                </item>
                </efficiencies>
            </info>
          </item>
          <item>
            <commod>uox_U</commod>
            <info>
              <buf_size>91600</buf_size>
              <efficiencies>
                <item>
                  <comp>U</comp> <eff>.998</eff>
                </item>
                </efficiencies>
            </info>
          </item>
        </streams>
      </Separations>
    </config>
  </facility>
```

Facility - Cycamore::Reactor

```
<facility>
  <!-- France -->
  <!-- PWR -->
  <name>FLAMANVILLE-1</name>
  <config>
    <Reactor>
      <fuel_inrecipes>  <val>uox_fuel_recipe</val>       <val>mox_fuel_recipe</val>       </fuel_inrecipes>
      <fuel_outrecipes> <val>uox_used_fuel_recipe</val> <val>mox_used_fuel_recipe</val>  </fuel_outrecipes>
      <fuel_incommods>  <val>uox</val>                  <val>mox</val>                   </fuel_incommods>
      <fuel_outcommods> <val>french_uox_waste</val>            <val>mox_waste</val>            </fuel_outcommods>
      <fuel_prefs>      <val>1.0</val>                  <val>3.0</val>                   </fuel_prefs>
      <cycle_time>18</cycle_time>
      <refuel_time>2</refuel_time>
      <assem_size>446</assem_size>
      <n_assem_core>257</n_assem_core>
      <n_assem_batch>86</n_assem_batch>
      <power_cap>1330</power_cap>
    </Reactor>
  </config>
</facility>
```

Region

```xml
<region>
  <name>Poland</name>
  <config>
    <NullRegion/>
  </config>
  <institution>
    <name>Poland_government</name>
    <config>
      <DeployInst>

        <prototypes>
          <val>CHOCZEWO</val>
          <val>NONAME</val>
        </prototypes>

        <build_times>
          <val>708</val>
          <val>780</val>
        </build_times>

        <n_build>
          <val>1</val>
          <val>1</val>
        </n_build>

        <lifetimes>
          <val>720</val>
          <val>720</val>
        </lifetimes>

      </DeployInst>
    </config>
  </institution>
</region>
```

```xml
<recipe>
  <name>natl_u_recipe</name>
  <basis>mass</basis>
  <nuclide>
    <id>U235</id>
    <comp>0.711</comp>
  </nuclide>
  <nuclide>
    <id>U238</id>
    <comp>99.289</comp>
  </nuclide>
</recipe>
```

Benefits

1. User can generate input file from database
   - Reactor Specifications
   - Facility deploy / decom times
   - Spent fuel recipe
2. Sensitivity study using external driver (e.g. RAVEN)
3. Simple automation / modification of input file

Output

Cyclus records all transactions between `Facilities` and other metrics unique to each archetype, such as:

- Cycamore::Reactor - Power generation per timestep
- Cycamore::Enrichment - SWU per timestep

Example Workflow

This workflow was used in the paper `Synergistic Spent Nuclear Fuel Dynamics Within the European Union` (in ANS 2017 Winter meeting, journal publication pending).

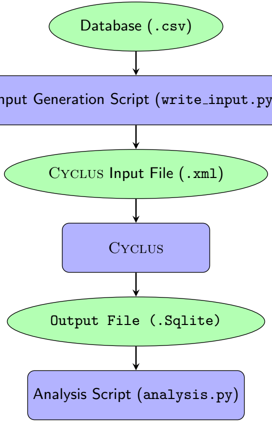


Figure 2: Green circles and blue boxes represent files and software processes, respectively, in the computational workflow.

CSV to Cyclus input file

1. Python script to parse through CSV file (reactor name, start / decom date, power output, etc)
2. Use Jinja template to construct input file (Python script fills curly brackets)

```
<facility>
   <!-- {{ country }} -->
   <!-- {{   type  }} -->
   <name>{{ reactor_name }}</name>
   <config>
     <Reactor>
       <fuel_inrecipes>  <val>uox_fuel_recipe</val>       </fuel_inrecipes>
       <fuel_outrecipes> <val>uox_used_fuel_recipe</val>  </fuel_outrecipes>
       <fuel_incommods>  <val>uox</val>                   </fuel_incommods>
       <fuel_outcommods> <val>uox_waste</val>             </fuel_outcommods>
       <fuel_prefs>      <val>1.0</val>                   </fuel_prefs>
       <cycle_time>18</cycle_time>
       <refuel_time>2</refuel_time>
       <assem_size>{{assem_size}}</assem_size>
       <n_assem_core>{{ n_assem_core}}</n_assem_core>
       <n_assem_batch>{{n_assem_batch}}</n_assem_batch>
       <power_cap>{{capacity}}</power_cap>
     </Reactor>
   </config>
 </facility>
```

Output analysis

- Python script to query and process output data
- Use Jupyter notebook to organize / visualize output

Output - regional analysis

- The user can separate analysis by regions
- Concept of children-parent: each `facility` has a parent `Institution`, and each `Institution` has a parent `Region`.

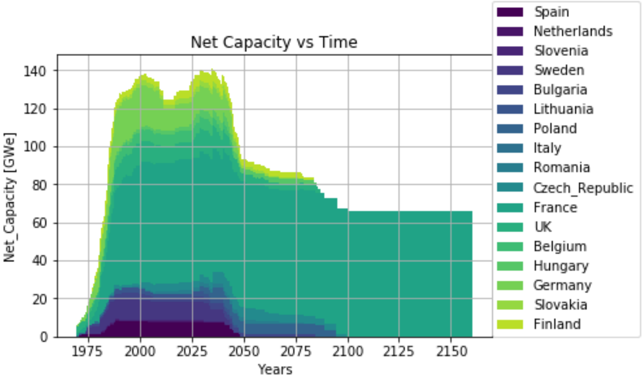


Figure 3: Power generation is separated by region.

- The user can separate analysis by regions
- Concept of children-parent: each `facility` has a parent `Institution`, and each `Institution` has a parent `Region`.

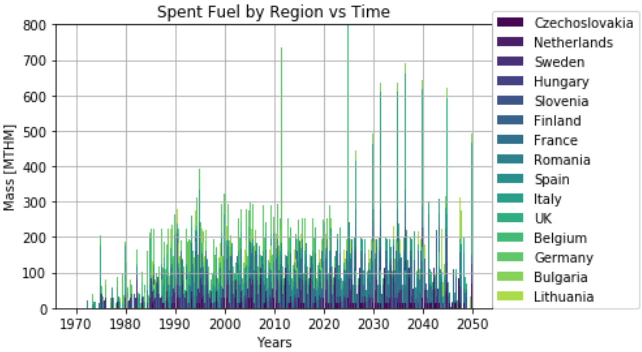


Figure 4: Waste output mass is separated by their origin region.

Output - Prototype analysis

- The user can separate analysis by prototype
- User can see how much `power` is from SFRs compared to PWRs.

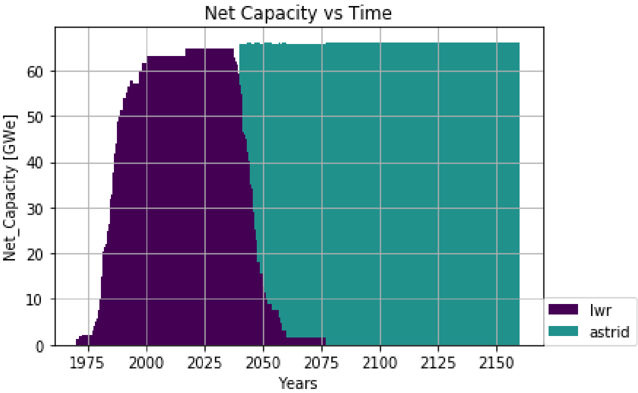


Figure 5: Power generation is separated by prototype (SFR, PWR).

Output - Prototype analysis

- The user can separate analysis by prototype
- User can see how much fuel is from which facility.

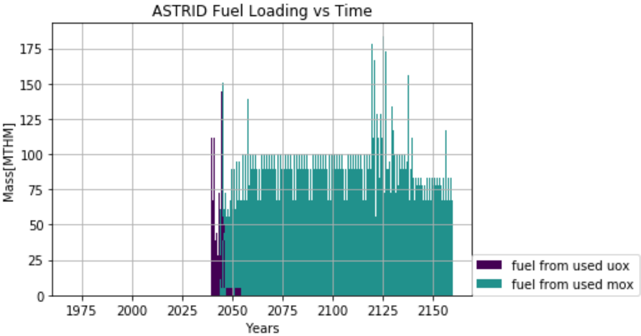


Figure 6: Fuel production is separated by production facility.

Sensitivity Study

Breeding ratio sensitivity study can be done by simply changing the SFR output fuel recipe in the input file.

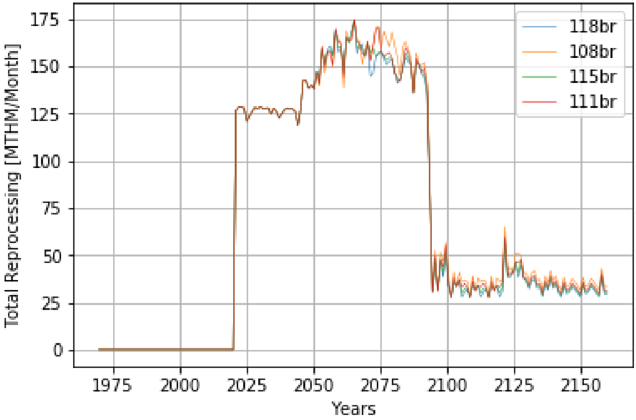


Figure 7: Breeding Ratio affect on total reprocessing.

Lifetime extension sensitivity study can be done by adding the lifetime of the pwrs and adjusting SFR deployment accordingly.

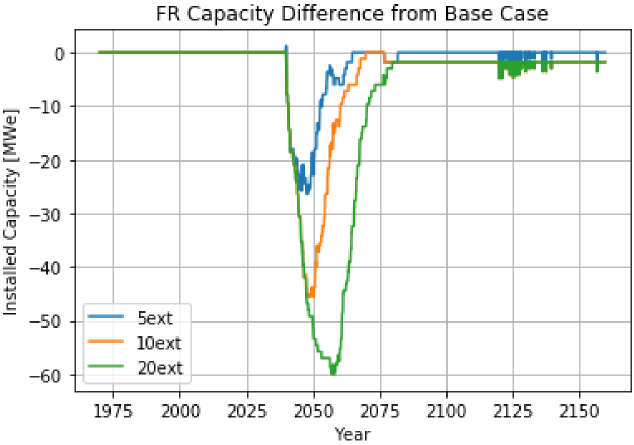


Figure 8: PWR lifetime extension affect on FR installed capacity.

Predicting the past - U.S

Work done by undergraduate researcher Gyutae Park at the University of Illinois - Urbana Champaign.

1. Import database to construct Cyclus simulation
2. ‘Predict the past’ - fuel usage, power generated
3. Demonstrate GIS capabilities of Cyclus

Example Workflow

Similar workflow has been used for this analysis study.

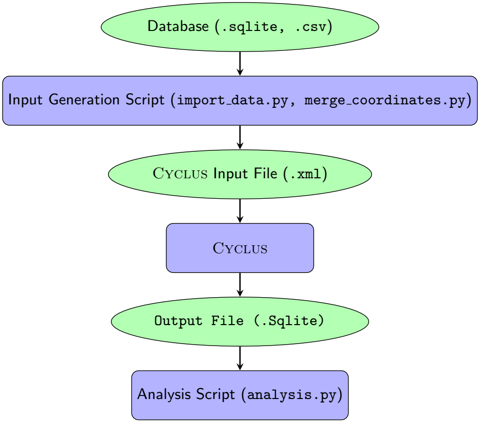


Figure 9: Green circles and blue boxes represent files and software processes, respectively, in the computational workflow.

Results - Fuel into reactors

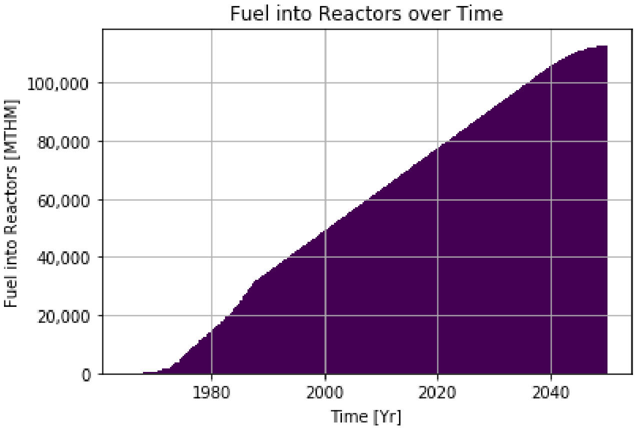


Figure 10: Cumulative fuel into U.S. reactors over time.

Results - Natural Uranium usage

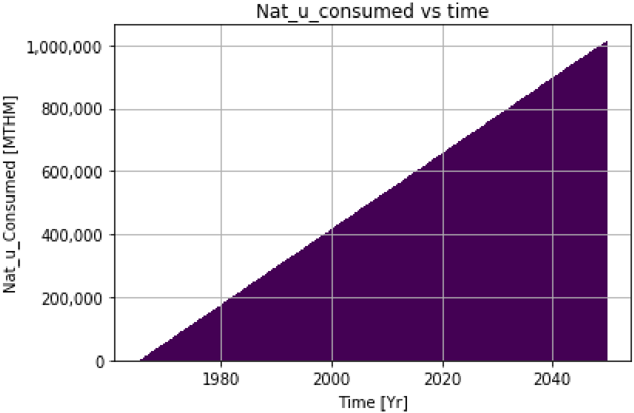


Figure 11: Cumulative natural uranium consumption in the U.S. over time.

Results - Power generated

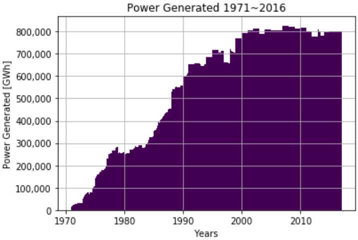


Figure 12: Nuclear Power generated simulated by Cyclus

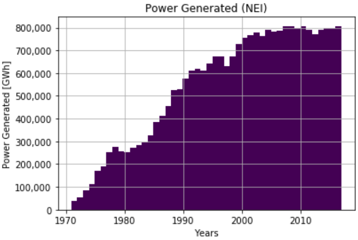


Figure 13: Nuclear power generation data from NEI [3]

[3] US Nuclear Generating Statistics. (n.d.). Retrieved from https://www.nei.org/Knowledge-Center/Nuclear-Statistics/US-Nuclear-Power-Plants/US-Nuclear-Generating-Statistics

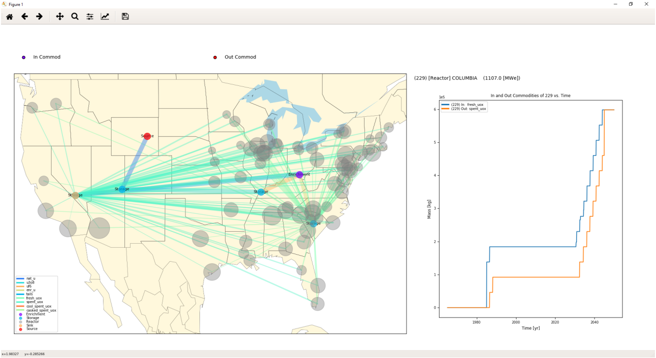


Figure 14: Interactive map of U.S. reactors and fuel cycle facilities. Lines show transactions between two facilities.

Conclusion

Cyclus is a performant, expanding fuel cycle simulator that holds promise for future applications. It demonstrated its capability to:

- 'Predict the past'
- Model transition scenarios
- Visualize important fuel cycle metrics

Future Work Ongoing

`Cycamore` is adequate for rough analyses, but more accurate modules or additional tools would increase analysis fidelity

- Dynamic archetype parameters (e.g. `refuel_time` changing in time or sampled from a distribution)
- In-module depletion (i.e. Using in-module SERPENT Reduced-order-model)
- Demand-driven deployment [4]

[4] NEUP 16-10512

Thank you.