Fundamental Concepts in the Cyclus Fuel Cycle Simulator Framework and Modeling Ecosystem

Kathryn D. Huff[1],

Robert W. Carlsen[2],

Robert Flanagan[3],

Matthew J. Gidden[2],

Arrielle C. Opotowsky[2],

Olzhas Rakhimov[2],

Anthony M. Scopatz[2],

Zach Welch[2],

Paul P.H. Wilson[2]

[1]*University of California - Berkeley, Department of Nuclear Engineering Berkeley, CA, 94720*

[2]*University of Wisconsin - Madison, Department of Nuclear Engineering and Engineering Physics, Madison, WI 53706*

[3]*University of Texas - Austin, Department of Mechanical Engineering, Nuclear and Radiation Engineering Program, Austin, TX*

Send proofs to: Kathryn D. Huff

huff@berkeley.edu

2150 Shattuck Ave., Suite 230, Berkeley, CA 94704

Number of Pages: ??

Number of Tables: ??

Number of Figures: ??

Keywords: fuel cycle, simulation, software

I INTRODUCTION

In nuclear fuel cycle simulation it is necessary to calculate and optimize nuclear process physics, facility deployment, material routing, etc.

Historically, nuclear fuel cycle simulators have addressed this with software ranging from spreadsheet driven flow calculators to full detail industrial models.

An alternative to these approaches is a dynamic, agent-based simulator with an open platform.

IA Background

Simulators drive R&D by calculating metrics that can be compared accross fuel cycles.

Previous simulators have been static or dynamic. Some have been discrete while others have been fleet-based. Some have captured regional behavior, material isotopics, transmutation detail, etc. Indeed, some tools have attempted agent-based functionality.

Historically, the national laboratories have driven development and regulated the use of their own tools (cite VISION, DANESS, NFCSim, etc.), while universities and industry have simultaneously developed tools of their own as well.

The Cyclus code arose out of the GENIUSv2 code. That software itself was the result of lessons-learned from an LDRD project at INL called GENIUSv1.

IB Motivation

Cyclus provides a sophisticated and modern, agent-based platform. Previous codes were lacking in a number of ways. Cyclus has a number of features that overcome those issues.

IB.ii Modular Architecture

Draw from and cite the "Open Architecture and Modular Paradigm" ANS summary.

We expect that there are a number of types of users and developers who will interact with the Cyclus

framework. Those people fall into a number of categories. All such people benefit from the modular architecture.

<modified-open-source collaboration diagram>

IB.ii Agent-based Paradigm

ABM is modern, sophisticated, true to reality, and it takes advantage of modern computing resources.

IB.ii Discrete Resource Tracking

Not all simulations can be successfully run with a fleet-based model that doesn't distinguish between discrete materials. Shadow fuel cycles are an example.

II METHODOLOGY AND IMPLEMENTATION

Only a modular, agent-based approach is capable of adequately solving system dynamics problems involving material routing, facility deployment, regional and institutional hierarchies.

IIA Framework Structure

Agent-based modeling is inherently object oriented.

The core of the simulator creates a set of key classes on which agent plugins are based. In addtion, a set of key tools are also provided, to enrich the API and provide a robust suite of behaviors for the developer.

<diagram of core, modules, toolkit, etc>

Agent plug-ins utilize the generic core API to interact with one another. Mainly they do this by trading resources.

<diagram of black box facilities>

IIB Cluster-Ready Software

Because this doesn't rely on PowerSim, GoldSim, Excel, or any other COTS Windows-based software, we can run singular instances of Cyclus on large linux machines that have lots more capability than hacked together Windows clusters.

IIC Dynamically loadable libraries

A key innovation that has previously not been implemented in nuclear fuel cycle simulators in the literature is implementing this generic API and modular architecture into a suite of dynamically loadable "plug-in" libraries.

This approach allows contribution to an ecosystem of libraries. The scientist-developer can therefore focus on generating a model within their sphere of expertise, while relying on the model contributions of others to fill in the other technologies.

This is implemented largely through a clean API and a modern build system.

IID Agent Interchangability

This approach enables "apples-to-apples" comparisons, something that has been sorely missing in previous simulators.

By making the facilities interchangeable, it combinatorically increases the number of scenarios that can be modeled. The user can choose to model just the front end or just the back end of a fuel cycle.

This is implemented with an API that focuses on treating the agents as black boxes.

IIE Region/Institution/Facility hierarchy

Not many fuel cycle simulators have this, it turns out. We used to implement this with inheritance. Now I'm not so sure. . . <insert recent knowledge here>.

IIF Discrete Object Tracking

Discrete facility and material tracking is more realistic an dmore

IIG Toolkit

The toolkit is a place for developers and users to contribute common-use tools for calculating metrics, managing process physics, handling data.

This was implemented by liberal use of namespaces and is populated with an introductory suite of useful tools.

IIH Cycamore

Cycamore contains a number of usefule facility models that are a mere base-set. Additional modules are needed for full fuel cycle simulation. However, simple fuel cycles can be generated with Cycamore and Cyclus alone.

<diagram of a possible Cycamore-only simulation>

III EXAMPLES

Current and past simulations with the Cyclus simulator have already demonstrated a range of uses.

IIIA Ecosystem

External modules have been (cite cyder, separations, streamblender, demandinst, commodconverter, etc.) and are being (cite brightlight, UTK module, etc.) developed for contribution to the cyclus ecosystem of models.

IIIB Simulations

Simulations have been run in the past and more are being run. FCO is an example.

IV VERIFICATION

Unfortunately, we can't wind forward the clock to compare our simulations to the futures that we are attempting to model.

However, Cyclus does implement a number of strategies for verification and validation of code.

IVA Coding Practices

Some routines are implemented specifically to ensure robustness.

IVB Testing Framework

We have reasonable test coverage and a code review system.

IVC Benchmarks

A simple transition scenario recently used for a code-to-code comparison within the national laboratories can demonstrate the ability for Cyclus to conform to externally defined specifications.

V CONCLUSIONS