Status of LLNL's GNDS C++ APIs

Presented to NEA/WPEC Subgroup 43

Bret Beck

April 2020

LLNL-PRES-XXXXXX

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Overview

- GIDI+
  - A suite of 5 code packages which together support access to GNDS data for transport codes
  - For Monte Carlo provides routines for sampling GNDS data
- Map file and classes
  - Organizes GNDS files into a library
  - Handled by GIDI
- PoPI – reads GNDS PoPs data/files
- GIDI - reads in GNDS files
- MCGIDI - extracts GNDS data from GIDI and samples data for Monte Carlo codes
- Other structures
- Building
- Future - to do
- Conclusion

GIDI+ available at https://github.com/LLNL/gidiplus
FUDGE available at https://github.com/LLNL/fudge

Initial design of GIDI+

- GIDI - General Interaction Data Interface
- Initial requirements
  - Ability to read a PoPs file: provided by the PoPI C++ API
  - Ability to read a GNDS file: provided by the GIDI C++ API
  - Easily access multi-group data for deterministic codes
    - Including summing multi-group data (e.g., total cross section from sum of reaction cross sections): provided by the GIDI C++ API
  - Ability to evaluate continuous energy data and sample as required by Monte Carlo transport codes : provided by the MCGIDI C++ API
    - Multi-group also but not for distribution data.
  - Ability to turn off some reactions: Implemented in GIDI which MCGIDI obeys

Initially did not include writing or being convenient for evaluator/developer of GNDS files

GIDI+

- 5 code packages developed at LLNL
  - 2 C code packages : these are also used in FUDGE
    - statusMessageReporting
      - Provides error handling for numericalFunctions
    - numericalFunctions
      - Object to handle 1d function
      - For example, supports addition, subtraction, multiplication of two 1d functions
  - 3 C++ code packages : described on later slides
    - PoPI
    - GIDI
    - MCGIDI
- Requires third party package pugixml (version 1.8)

Available to download from https://github.com/LLNL/gidiplus

Contents of GIDI+ directory

```
statusMessageReporting/
numericalFunctions/
PoPI/
GIDI/
MCGIDI/
Doc/
Misc/                          # Download pugixml-1.8.zip here
Scripts/
include/
lib/
COPYRIGHT
LICENSE
Makefile
NOTICE
README.md
```

Example contents of PoPs, GIDI and MCGIDI directories

```
Doc/                    # Doxygen documentation – some exists
Examples/               # Examples used in documentation; currently only 1 in GIDI
Makefile
Src/                    # C++ *.hpp and *.cpp source files
Test/                   # Contains many test run with “make check”
bin/                    # Contains a few useful executable for examining GNDS files
include/                # All needed *.hpp file are put here during build
lib/                    # All needed lib* files are put here during build
```

Map files: building a nuclear data library from GNDS file

El protare

- A GNDS file represents a projectile hitting a target for one evaluation
  - Examples
    - n + O16 for evaluation ENDF/B-VIII.0
    - p + Li7 for evaluation ENDL2009.4
- I call a GNDS file a protare
  - PROjectile + TARget for an Evaluation
  - This is called a reactionSuite in GNDS
- In GIDI and MCGIDI the protare is represented by Protare classes
  - Yes, it is class**es**, even within GIDI as will be described later
  - The 'el' (library) part will be described later also
  - No, protare is not a Spanish word as far as I can tell

Nuclear data libraries

- Examples are ENDF/B-VIII.0, JEFF 3.3, ENDL2009.4
- Consist of a collection of protares
- Map file
  - LLNL has defined a map file to create a library from a collection of protares
  - This is similar to the 'directory' section of a MCNP xsdir file
    - Example line from an xsdir file

```
1001.60c 0.999170 endf6011 0 1 1 3484 0 0 2.5300E-08
```

    - In a map file this is

```
<protare projectile="n" target="H1"
   evaluation="ENDF/B-VIII.0" path="n-001_H_001.xml"
   interaction="nuclear"/>
```

A map file contains many protare nodes (and a few other types) to make a library.

Contents of a Map file

- Map node has two attributes
  - library: name of the library
  - format: specifies the format/structure of the map file
    - Currently, LLNL working on 0.2

```
<map library="ENDF/B-VIII.0" format="0.2">
```

- Map node has three types of child nodes
  - protare: specifies a protare and its path
  - TNSL: specifies a protare that contains only TNSL data and a reference to the protare to use when outside the energy/temperature domain of the TNSL data
  - import: specifies a path to another map file
    - Map files can be nested

Map file example

```
<map library="neutrons" format="0.2">
  <protare projectile="n" target="H1" evaluation="ENDF/B-8.0”
        path="n-001_H_001.xml" interaction="nuclear"/>
  <TNSL projectile="n" target="HinCH2" evaluation="ENDF/B-8.0”
        path="tsl-HinCH2.xml”
        standardTarget="H1" standardEvaluation="ENDF/B-8.0"/>
  <protare projectile="n" target="O16" evaluation="ENDF/B-7.1”
        path="n-008_O_016.xml" interaction="nuclear"/>
  <protare projectile="n" target="Al27" evaluation="ENDF/B-8.0”
        path="n-013_Al_027.xml" interaction="nuclear"/>
  <TNSL projectile="n" target="tnsl-Al27" evaluation="ENDF/B-8.0”
        path="tsl-013_Al_027.xml”
        standardTarget="Al27" standardEvaluation="ENDF/B-8.0"/>
  <protare projectile="n" target="Fe56" evaluation="ENDF/B-7.1”
        path="n-026_Fe_056.xml" interaction="nuclear"/>
  <protare projectile="n" target="Th227" evaluation="JENDL-7.1”
        path="n-090_Th_227.xml" interaction="nuclear"/>
  <import path="neutrons/all.map”/></map>
```

GIDI map support

- The class GIDI::Map::Map
  - Supports reading a map file
  - Supports three types of protares
    - Standard protare:
      - A single GNDS file
      - ProtareSingle class in GIDI and MCGIDI
    - TNSL protare
      - Contains two GNDS files
        - A ProtareSingle with TNSL only data
        - The standard ProtareSingle associated with TNSL protare
      - ProtareTNSL class in GIDI and MCGIDI
    - Composite protare
      - A list of ProtareSingle's
      - Currently used to treat photo-atomic and photo-nuclear as if they are one protare
      - Could be used to treat isotopes of an element as a "natural" element
      - ProtareComposite class in GIDI and MCGIDI

Examples of using a Map instance

- Creating a Map instance

```
#include <GIDI.hpp>

PoPI::Database pops( "../Test/pops.xml" );
GIDI::Map::Map map( "../Test/all.map", pops );
```

- Getting protares for ”n + O16” and “photon + O16”

```
GIDI::Construction::Settings construction( GIDI::Construction::ParseMode::all,
                                           GIDI::Construction::PhotoMode::nuclearAndAtomic );

GIDI::Protare *protare1 = map.protare( construction, pops, "n", "O16" );
GIDI::Protare *protare2 = map.protare( construction, pops, ”photon", "O16" );
```

- Definition of Map::protare function

```
GIDI::Protare *protare(   Construction::Settings const &a_construction,
                          PoPI::Database const &a_pops,
                          std::string const &a_projectileID,
                          std::string const &a_targetID,
                          std::string const &a_library = "",
                          std::string const &a_evaluation = "", ... );
```

The GIDI package

GIDI

- General Interaction Data Interface
- Reads in a GNDS file
- Has three protare classes that inherit from the pure-virtual Protare class
  - ProtareSingle: represents a single GNDS file
  - ProtareComposite: Contains a list of ProtareSingle instances
    - For multi-group data, sums results from each ProtareSingle instance
    - Currently used by Map instance to treat photo-atomic and photo-nuclear for a specified target as one protare
  - ProtareTNSL: Contains two ProtareSingle instances
    - Gets data from the one of the protares based on projectile energy and target temperature

In general, user should access GIDI protare through a Map instance rather than instantiating themselves.

GIDI continued

- Access nearly all data in a Protare instance
  - Does not yet support resonance and covariance data
- Supports multiple temperature data in a GNDS file
- Convenient functions for getting multi-group data
- Can set reactions as active or inactive
  - Up to codes to obey active state
  - Multi-group functions obey active state
  - MCGIDI obeys active state

Some GIDI functions

- numberOfReactions( ) and reaction(index)
  - Number of reactions in a ProtareComposite (and ProtareTNSL) is the sum of all reactions over all embedded ProtareSingles
- numberOfOrphanProducts( ) and orphanProduct(index)
- multiGroup*
  - Returns either a GIDI::Vector or GIDI::Matrix instance
    - Vector and Matrix classes support addition as expected
    - May rename to GIDI::MGVector or GIDI::MGMatrix
- GIDI::Styles::TemperatureInfos temperatures( )
  - Returns information (i.e., labels) useful to for getting access to data for one of the temperatures

Examples of accessing data (ideally)

- Get ‘evaluated’ style cross section for 3rd reaction

```
GIDI::Reaction *reaction = protare->reaction( 2 );
GIDI::Functions::Function1d *crossSection = reaction->crossSection( ).get( ‘eval’ );
```

- GIDI::ProtareSingle vs. FUDGE
  - GIDI::ProtareSingle

```
GIDI::Reaction *reaction = protare->reactions( ).get( 2 );
GIDI::Functions::Function1d *crossSection = reaction->crossSection( ).get( ‘eval’ );
```

  - FUDGE

```
reaction = protare.reactions[2];
crossSection = reaction.crossSection[‘eval’];
```

Many objects in GNDS are list/dictionary like which GIDI and FUDGE treat as such.

GIDI::Protare temperatures function

- Styles::TemperatureInfos temperatures( );
- typedef std::vector<Styles::TemperatureInfo> TemperatureInfos;
  - A list of Styles::TemperatureInfo instances

```
class TemperatureInfo {

    private:
        PhysicalQuantity m_temperature;      // The temperature for this TemperatureInfo data.

        std::string m_heatedCrossSection;    // The label for the heatedCrossSection data.
        std::string m_griddedCrossSection;   // The label for the griddedCrossSection data.
        std::string m_URR_probabilityTables; // The label for the URR_probabilityTables data.
        std::string m_heatedMultiGroup;      // The label for the heatedMultiGroup data.
        std::string m_SnElasticUpScatter;    // The label for the SnElasticUpScatter data.

        PhysicalQuantity const &temperature( ) const { return( m_temperature ); }
        std::string const &heatedMultiGroup( ) const { return( m_heatedMultiGroup ); }
...
};
```

```
GIDI::Styles::TemperatureInfos temperatures = protare->temperatures( );
```

Examples of accessing multi-group data

- Getting total multi-group cross section for a protare

```
GIDI::Styles::TemperatureInfos temperatures = protare->temperatures( );

GIDI::Transporting::MG settings( protare->projectile( ).ID( ),
                GIDI::Transporting::Mode::multiGroup,
                GIDI::Transporting::DelayedNeutrons::on );

GIDI::Vector crossSection = protare->multiGroupCrossSection( settings, temperatures[0] );
```

- Getting multi-group cross section for a reaction

```
GIDI::Reaction const *reaction = protare->reaction( index );

GIDI::Vector crossSection = reaction->multiGroupCrossSection( settings, temperatures[0] );
```

For total can always skip some reactions by setting them to inactive.

Particles class

- Tells GIDI and MCGIDI what particles are being transported
  - Stores a list of transportable “Particle” classes
    - Each Particle class has multi-group and flux info for multi-group collapsing
  - Used for multi-group collapsing, calculating energy deposition
  - Used by MCGIDI to throw away distributions, etc. for particles that are not being transported
- Example

```
GIDI::Fluxes fluxes( "../fluxes.xml" );
GIDI::Groups mgs( "../groups.xml" );
GIDI::Transporting::Particles particles;

GIDI::Transporting::Particle neutron( "n", mgs["LLNL_gid_4"], fluxes["flux_1"] );
particles.add( neutron );

GIDI::Transporting::Particle photon( "photon", mgs["LLNL_gid_4"], fluxes["flux_12"] )
particles.add( photon );
```

Not quite the current implementation but will be soon.

Using a Particles instance

- Getting multi-group, energy deposition (like KERMA)

```
depositionEnergy = protare->multiGroupDepositionEnergy( settings, temperatures[0], particles );
```

- Collapsing multi-group data
  - Works for GIDI::Vector and GIDI::Matrix

```
collapsed = GIDI::collapse( uncollapsed, settings, particles, 0, "photon" );
```

The MCGIDI package

MCGIDI

- Has three protare classes equivalent to the three GIDI protare classes
- MCGIDI::ProtareSingle extracts data needed from the GIDI::ProtareSingle instance and puts it into a better form for lookup and sampling
- Uses integers instead of PoPs' ids (i.e., std::string) to identify particles
  - Integers are faster to compare than std::strings
  - Get unique integer ids from PoPI instance
- Use function MCGIDI::protareFromGIDIProtare to get a MCGIDI protare from a GIDI protare
  - Like Map::protare, it understands the three types of protares
  - Takes a GIDI::Protare instance as its first argument

```
MCGIDI::Protare *MCProtare = MCGIDI::protareFromGIDIProtare( *protare, pops, MC,
          particles, domainHash, temperatures, reactionsToExclude );
```

Lookup and sampling

- Getting a protare's cross section given a projectile's energy and target's temperature

```
double crossSection = MCProtare->crossSection( URR_protare_infos,
                                hashIndex, temperature, energy );
```

- Sampling a reaction from a protare

```
int reactionIndex = MCProtare->sampleReaction( URR_protare_infos, hashIndex,
                          temperature, energy, crossSection, float64RNG64, rngState );
```

Sampling a reaction's outgoing particles

```
reaction->sampleProducts( MCProtare, energy, input, float64RNG64, rngState, products );
```

Other structures building, future/to do and conclusion

Other structures and files

- On LLNL super computers the following exists

```
ls -1 /usr/gapps/data/nuclear/common/
    pops.xml                # Contains particle definitions
    LLNL_alias.xml          # Aliases to map ZAs to PoPs ids (e.g., 8016 for O16)
    metastables_alias.xml   # Defines meta-stable aliases
    fluxes.xml              # Standard set of fluxes
    groups.xml              # Standard set of multi-group definitions
```

- Multi-group definitions
  - Stores a list of GNDS multi-group nodes
  - That is, a list of boundary definitions for each multi-group id
- Flux definitions
  - Stores a list of 3d-functions, each representing f(T, E, mu)
  - This is a flux as a function of temperature (T), projectile energy (E) and cosine of angle (mu)

Building GIDI+

- Should only need to specify C, CXX, CFLAGS, CXXFLAGS and PREFIX macros to build with Makefile

```
make C=gcc CFLAGS="-g –O0" CXX=g++ PREFIX=`pwd`/test_install/gnu
```

- Builds all packages including pugixml
- Main Makefile targets
  - default : builds libraries for all packages
  - install : does default target and then puts header files in $(PREFIX)/include and libraries in $(PREFIX)/lib
  - realclean : removes stuff built with 'default' target

To do

- Support hybrid files
  - using XML for all nodes except for values nodes
  - Values nodes are mainly a list of float64s that consume 90% of the access time when stored in ASCII (conversion of string to binary)
  - On disk, store data in values nodes in a binary representation
  - HAPI, developed by Caleb, supports this but not completely implemented into GIDI
- Beyond the initial requirement
  - Add support for writing GNDS file
  - Add convenient functions for evaluator, etc. to develop a GNDS file
  - Add support for resonance and covariance data
- Run time processing
  - Stretch goals
  - Examples
    - Heating on the fly
    - Multi-grouping on the fly

Conclusion

- GIDI+ released
  - Reads GNDS/PoPs files
  - Supports multi-group summing for transport codes (via GIDI)
  - Supports Monte Carlo continuous energy lookup and sampling
  - MIT License
  - Only reads GNDS 1.10
    - LLNL development version with GNDS format proposals already implemented
    - Current FUDGE produces GNDS 1.10
  - Available at https://github.com/LLNL/gidiplus
- FUDGE release
  - Available at https://github.com/LLNL/fudge
  - We need to update this as this version only supports GNDS 1.9
  - Supports map files

Best to way for next release.

What is the reaction?

67 2 0 0 0 0 328 3650