FSML: Fusion Simulation Markup Language for Interoperability of Data and Analysis Tools
Svetlana Shasharina and Chuang Li

GRIDL: High-Performance and Distributed Interactive Data Language (time permitting…)
Svetlana Shasharina, Ovsei Volberg, Peter Stoltz and Seth Veitzer

Tech-X Corporation (DOE/SBIR)
CLADE 2005
Overview

• FSML
  – Background
  – Motivation
  – Design
  – Status

• GRIDL
  – Motivation
  – Design of GRIDL
  – Status
The nuclear fusion produces energy:

\[ D + T \rightarrow (\text{He}^4 + 3.53 \text{ MeV}) + (n + 14.06 \text{ MeV}) \]

\[ E_n = 14 \text{ MeV} \]
\[ E_{\alpha} = 3.5 \text{ MeV} \]

Collected at walls
Deposited in plasma

Don Batchelor’s (ORNL) figure
Heat (to overcome repulsion) and hold particles in magnetic traps

- We heat the particles so that the average energy is ≈ 100,000,000F (plasma)
- Then we should contain plasma for many reactions to happen

Don Batchelor’s (ORNL) figures
Heterogeneous settings impede integrated modeling and data visualization and analysis in fusion and plasma physics

- Multiple data output formats used in modeling codes
  - HDF5 (M3D, VORPAL, NIMROD)
  - NetCDF (parallel version is gaining popularity)
  - MDSplus (experiment, NIMROD, TRANSP)

- Different data organization even within one format within one type of physics
  - Different presentation of the same variable (potential vs. vector)
  - Different discretization
  - Different names (node-0 vs temperature)
  - Different units
  - Different file structures (nodes, attributes)

- Multiple data analysis and data visualization tools
  - SCIRun (University of Utah)
  - AVS/Express (Advanced Visual Systems)
  - IDL (Research Systems Inc)
  - OpenDX (open source)
MxN solution does not scale: need custom solution to analyze data from each code for each analysis tool.
More scalable and maintainable solution is to provide a translation layer for conceptually similar codes.
Data from different codes describing the same physics is conceptually the same: allowing domain vocabulary using XML

- Concise (minimal but full) formulation of some fusion domain data and ways to transform between representations
  - A plus for future integrated modeling (big thing in the fusion community for the international device ITER to be built in France)
- XML to describe the domain data
  - XML is proven interoperability tool
  - Schemas can impose what should be present and how many times
  - Instances are validated against schema
  - Multiple APIs (C, C++, Java)
  - Many XML tools exist, wide industry support, Web Services
  - All other tools translate to XML and back (UML, Interface Definition Language, SIDL).
- Should we move to ontology?
- Need to narrow the fusion problem: we chose fusion MHD (magnetohydrodynamics) as our domain and a couple of tools for MHD data analysis and viz. Contradicts the original name (FSML). Might rename to MHDMML 😊
MHD codes (NIMROD and M3D) as foundation for FSML

- Treat plasma as charged fluids. Fluids are described by velocities and fields (E and B) on a grid.
- Solve PDE (extension of non-linear diffusion-type equations for many variables and multiple sources and sinks) and show evolution in time.
- Store data in HDF5 (hierarchical binary format suited for multidimensional arrays). Information: values of velocities and fields on a grid organized by time slices.
- Attributes and data organization are different in every MHD code using HDF5.
- Use different “networks” of SCIRun and AVS/Express.
NIMROD uses finite difference in poloidal plane and Fourier in toroidal direction
Unstructured mesh needs more information

Schematic of the poloidal plane of M3D mesh
SciRun for NIMROD shows heat deposition, temperature isosurfaces, and field lines
AVS/Express shows magnetic surfaces and MHD activity modeled by M3D
MHD data has 3 levels

- Syntax (numerical types) is taken care of HDF5 itself
- Semantic MHD schema will include the unified set of MHD variables (temperature, pressure, velocity, current density, magnetic field, and particles density for all species etc) and geometry information
- Content metadata
  - simulation and publication relevant data (authors, platforms, compilers, goals, notes)
  - connectivity for multiple distributed files (large data might get split and end up at different sites)
  - abbreviated descriptions of data
Data analysis and visualization also need

• Uniform mesh descriptions (dimensionality, nodes and connectivity)
  – Structured and unstructured meshes
  – Optimization when possible (for structured meshes)
  – Problem is of interest to TSTT, the Terascale Simulation Tools and Technology center (MOAB C++ library from Sandia National Laboratory)

• Data in arbitrary point (not just on mesh)
  – Interpolate in accordance with discretization of the code (not linear interpolation used currently)
  – Problem of interest to TSTT (the Field Library)
Proposed solution for MHD: FSML

- Study MHD codes and their outputs
  - M3D, NIMROD and a European code to be determined
- Develop XML schema for fundamental MHD variables
- Add XML tags for other common content
- C++ API based on the schema and implemented based on XML instances for access of native HDF5 data
- C++ API for uniform access of mesh data (structured and unstructured)
- C++ API for data interpolation for MHD data
- SCIRun and AVS/Express modules wrapping C++ APIs
- Component representation for all elements (for future integrated modeling)
FSML Architecture allows for new MHD codes and new analysis tools
FSML has been prototyped

- Partial XML schema (temperature, magnetic field, time, geometry) and C++ parsing libraries for the variables
- AVS/Express modules to do 3D visualization
XML schema included several common variables and geometry data
XML instances describe particular codes

```xml
<?xml version="1.0"?>
<a:FSML xmlns:a="FSML"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="FSML FSML.xsd">
  <SyntacticMetaData>
    <Hdf5 application="M3D">
      <Structure name="/" instances="1">
        <Field name="time">
          <Time name="m3d-time", unit="second"/>
        </Field>
      </Structure>
      <Structure name="time_node_data[0]" instances="1">
        <Field name="node_data[9]" instances="1">
          <Magnetic name="m3d-magnetic" unit="Tesla"/>
        </Field>
      </Structure>
    </Hdf5>
  </SyntacticMetaData>
</a:FSML>
```
AVS/Express and SCIRun use uniform API to access all data.
Future is great

• Funding for 2 year project (2 FTE per year) just received from DOE/SBIR (thanks!)
• Proceed as described above
GRIDL: High-Performance and Distributed Interactive Data Language

Svetlana Shasharina, Ovsei Volberg, Peter Stoltz and Seth Veitzer

Tech-X Corporation

See poster at HPDC 2005, July 25, 2005
Overview

- Interactive Data Language
- Design of parallel IDL on a grid
- Design of IDL clients for Web/Grid Service
- Status
- Conclusions
Interactive Data Language (IDL) is a powerful visualization and analysis tool

- 4GL: scripting and high-level (built on C)
- Well-suited for N-dimensional arrays
- Interworks with C/C++, Fortran, and Java
- Well established in scientific community
  - Fusion community uses IDL interface to access and analyze experimental data
  - Plasma physics simulations (VORPAL: 3D code) make 2D plots, 3D objects and movies using IDL
  - Volumetric MRI data (Los Alamos biophysics and co-registration codes at Ohio Kettering center)
  - Earth sciences
  - Military applications
Scientific computing is parallel and distributed

- 3D simulations are large and need multiple processors (possibly distributed): high-performance in a regular sense and on a grid
- Data (computational and experimental) is distributed and large (terabytes produced in remote experiments and simulations)
- Data analysis is distributed and massive
- Many levels of parallelisms and interactivity:
  - Some directives need to be parallel
  - Some should happen on one processor in an interactive mode (finalization of data analysis)
Solutions (high-performance computing and Grids) exist

- **Grids:**
  - **Globus:**
    - GT3 (and pre GT3): secure running remote jobs
    - GT4: Web Services for scientific computing
  - **Web Services:**
    - Distributed client-server using Web technologies

- **MPI:** parallel computing
- **MPICH-G2:** MPI on a grid

**Grids and MPI have not addressed 4GLs and IDL in particular**
GRIDL: merging parallel distributed computing with IDL

- Parallel IDL
  - Running parallel IDL applications on clusters, supercomputers and grids
- IDL clients for Web Services
- 6 months of prototype work funded by DOE so far
Parallel IDL uses external C

- Dynamically Loadable Modules allow IDL to call external C functions by wrapping them.
- Wrapping C implementation of MPI exposes MPI in IDL
Parallel IDL on a Grid design uses external C and MPICH-G2

- **MPICH-G2**: MPICH using Globus underneath
- **Wrapping MPICH** now allows to run parallel IDL on a grid
IDL client wraps generated stub (C or Java)
Status

• DLMaker: prototype tool for automation of IDL wrapping of generic external C code
• Testing parallel IDL on a simple grid
• Playing with gSoap (wrapping of C++ clients of gSoap Web Services into IDL)
• Playing with Globus (wrapping Java clients of GT3.2 Web Services using Java-IDL bridge)
DLMaker works for simple types

- Input: .h file with C signatures
  
  ```
  extern "C" {
    void pro(double, int, float);
    int func(int, float);
  }
  ```

- Output: what is needed for exposing these C functions into IDL (C wrapper, registration code, DLM description). Allows:

  ```
  pro testmodule
  pro, 2, 3, 1
  y = func3(1, 3)
  end
  ```
Need more sophisticated DLMaker capable of:

- “In” and “out” variables with correct memory allocation (need extra information in the input file similar to CORBA’s Interface Definition Language or SIDL)
- IDL-type allocations of temporaries
- Produce IDL keywords (some arguments should go there: types)
- Specializing to particular web services compiler (wrappers should deal with return variables and static variables correctly – WS-compiler specific!)
- So far, we hand-wrapped C functions used in MPI and Web Services examples
Simple MPI wrapper can be produced by our DLMaker

```c
static IDL_VPTR IDL_MPI_COMM_SIZE
(int argc, IDL_VPTR argv[]){
    IDL_VPTR tmp = IDL_Gettmp();
    int size;
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    tmp->type = IDL_TYP_LONG;
    tmp->value.l = size;
    return tmp;
}
```
Parallel IDL looks like MPI then

```idl
pro testmodule, x
  rank=mpi_comm_rank()
  nproc=mpi_comm_size()
  help, rank
  help, nproc
end
```
Parallel IDL needs a C driver to do `mpirun`

```cxx
// startMPI.cxx
int main(int argc, char** argv) {
    int errorValue;
    MPI_Init(&argc,&argv);
    errorValue = IDL_Init(IDL_INIT_QUIET, &argc, argv);
    errorValue = IDL_ExecuteStr("testmodule, 1");
    MPI_Finalize();
    errorValue = IDL_Cleanup(0);
    return errorValue;
}

mpirun -np 4 -v startMPI
```
LAN grid with 2 dual processors nodes
(MPICH-G2 v. 1.2.6 and Globus 3.2, gcc3.43 and IDL v.6.1)

Speedup is linear
IDL clients for gSoap Web Service needed hand-wrapping

Interface:

```c
int wslidlhello__simproc1(int x, struct wslidlhello__simproc1Response {} *out);
```

Wrapper:

```c
void IDL_simpro1(int argc, IDL_VPTR* argv){
    int x = argv[0]->value.i;
    // gSoap runtime environment initialized
    struct soap soap;
    soap_init(&soap);
    // Call the method
    soap_call_wslidlhello__simproc(&soap,
        "grid.txcorp.com:18084", NULL, x, 0);
    // Clean up
    soap_end(&soap);
}
```
Working with Java clients and using Java-IDL bridge was straightforward

- GT3.2
- IDL 6.1 (or higher)
- Set paths as needed
- IDL client mirroring Java client
Java client:

```java
public class Client{
    public static void main(String[] args){
        // Get command-line arguments
        URL GSH = new java.net.URL(args[0]);
        int a = Integer.parseInt(args[1]);
        // Get a reference to the MathService instance
        MathServiceGridLocator mathServiceLocator = new MathServiceGridLocator();
        MathPortType math = mathServiceLocator.getMathServicePort(GSH);
        // Call remote method 'add'
        math.add(a);
        // Get current value through remote method 'getValue'
        int value = math.getValue();
    }
}
```
IDL client creates java objects and uses the bridge to delegate to them:

```idl
pro CLIENT
    ; URL for the service
    gsh = OBJ_NEW("IDLJavaObject$URL", "java.net.URL", 
                   "http://64.240.154.9:8090/ogsa/services/MathService")
    ; Service Locator
    mathLoc = OBJ_NEW("IDLJavaObject$LOC", 
                      "com.txcorp.stubs.MathService.service.MathServiceGridLocator")
    ; Get the port using service locator and service handle
    math = mathLoc->getMathServicePort(gsh)
    ; Invoke the method
    math->add, 7
    res = math->getValue()
    ; Clean
    OBJ_DESTROY, gsh
    OBJ_DESTROY, mathLoc
end
```
Conclusions

• Need more powerful and specialized tool for generation of IDL wrappers from C descriptions (for MPI and particular C Web Services).
• Need to wrap more MPI routines using the tool
• GT4 working with MPICH-g2
• Could use IDL-Java bridge to create IDL clients for Java Web Services. Works really well.
• Other 4GLs?
• Currently need new funding :-(