Four Parts:

- **Part 1: STK-Mesh Domain Model**
  - Comprehensive conceptual overview; no code

- **Part 2: STK-Mesh Computations**
  - How to perform computations; with code snippets

- **Part 3: STK-Mesh Modifications**
  - How to modify a mesh; with code snippets

- **Part 4: SIERRA Toolkit – beyond STK-Mesh**
  - Other modules / library components
STK-Mesh Domain Model

SIAM Conference on Computational Science and Engineering
February 28, 2011

H. Carter Edwards
Sandia National Laboratory
What is a *Domain Model*?

- A formal expression of, and ubiquitous language for,
  - Conceptual context
  - Software architecture
  - Software requirements framed within the conceptual context and software architecture context

- “Domain-Driven Design; Tackling Complexity in the Heart of Software” by Eric Evans
  - I highly recommend to developers of non-trivial CS&E software
  - Systematic and iterative approach to modeling a domain
  - Emphasis on managing complexity and mapping to software
The Challenging STK-Mesh Domain

• An Unstructured Mesh
  – “Weblike pattern or construction that fills a spatial domain $\Omega$”
  – A discretization of a spatial domain $\Omega$
  – Filled with arbitrary elemental subdomains; e.g., polyhedrons

• Supporting Computations
  – Simulations with multiphysics / heterogeneous phenomena
  – E.g., numerical solution of PDEs defined on the domain $\Omega$
  – Computational data associated with entities of the discretization

• That use Advanced Capabilities and Algorithms
  – Massively parallel and hybrid parallel computations
  – Solution strategies that adapt the computations & discretization
Modular, Layered Architecture

Focus of this talk: **Generic Computational Mesh Library (STK-Mesh/base)**

- Supporting Parallel Utilities
- Finite Element Specific Computational Mesh Library (STK-Mesh/fem)
- Finite Element Local-Operators
- Finite Element Shapes & node ordering (Trilinos/Shards)

Application Code
Computational Mesh Database

• Computational Mesh
  – Discretization data; e.g., nodes, elements, connectivity
  – Computation data; e.g., variables of the PDEs / models

• Mesh Database Bulk-data
  – Discretization and computation data of the problem to be solved
  – Size is proportional to the granularity of the discretization
  – Must be parallel-distributed for scalable computations

• Mesh Database Meta-data
  – Description of the bulk-data (a.k.a., the database’s schema)
  – Size is proportional to the complexity of the discretization
  – Assumed to be parallel-duplicated for simplicity
Mesh Bulk-Data: Discretization

- **Mesh Entity:** $entity^{Rank}_{GlobalID}$
  - The fundamental, atomic units of the discretization
  - E.g., finite element method’s nodes, edges, faces, and elements
  - Ranking within the discretization; e.g., node < element
  - Globally unique and persistent identifier
    - Across parallel-distributed memory space
    - For the lifetime of the entity
    - Fully ordered
  - Unique: (Rank, Global-ID)
Mesh Bulk-Data: Discretization

- **Mesh Entity Relation:** \((entity^J_a, entity^K_b, relationID)\)
  - Directed from higher to lower ranking entity
  - Rank \(J > K\) : \(entity^J_a \rightarrow entity^K_b\); required \(J \neq K\)
  - E.g., an element is defined to be higher ranking than a node
  - Uniqueness when \(J > K\): \((entity^J_a, K, relationID) \rightarrow entity^K_b\)
    - … or does not exist
Mesh Bulk-Data: Closure

• Directed Acyclic Graph (DAG)
  – Entities are nodes of the graph
  – Relations are directed edges of the graph
  – Acyclic: directed relations higher ➔ lower rank
  – Concept of “Closure”

• Closure of a Mesh Entity: $\text{entity}_a^J$
  – Collection of entities and relations
    • Reachable from that entity
    • Following directed relations
  – E.g., an element, its nodes, and element ➔ node relations
  – Assumption: The totality of computations performed on a given mesh entity require access to the closure of that mesh entity
Mesh Bulk-Data: Parallel Distribution

- Parallel distribution of mesh entities and relations
  - Every mesh entity is uniquely owned by a parallel process
  - *Closure* of an owned mesh entity must reside on that process
  - This requires duplication (sharing) of mesh entities
  - \( \overline{entity_a^J \cap entity_b^J} \) is shared between the owning processes
Mesh Bulk-Data: Parallel Distribution

• One Layer Ghosting (a.k.a., Ghosting Aura)
  – If a mesh entity is shared by a process
  – Is in the closure of another mesh entity: $\text{entity}_b^K \in \text{entity}_a^J$
  – Then that closure is also duplicated on the process
  – E.g., the closure of the elements connected to a shared node are ghosted on all processes on which the node is shared
  – Invaluable for neighborhood / patch accessing algorithms
Mesh Meta Data: Defining Subsets

• Multiphysics and Heterogeneity
  – Different models and computations in different subdomains
  – Heterogeneous discretizations: shapes, polynomial degree, ...

• Mesh Part: $Part_A$
  – Define subdomains for different computations
  – Define collections of mesh entities with the same discretization
  – Define supersets of parts
    $$Part_X = Part_A \cup Part_B \cup Part_C$$
  – # Mesh Parts depends upon heterogeneity (complexity) of the domain and computations
Mesh Bulk-Data: Part Membership

• A Mesh Entity is a Member of one or more Mesh Parts
  – \( \text{entity}_a^J \in Part_A \cap Part_C \cap Part_F \)
  – Always a member of the universal part: \( Part_\Omega \)

• Mesh Part Membership may be Induced
  – If \( \text{entity}_a^J \in Part_A \) and \( \text{entity}_a^J \rightarrow \text{entity}_b^K \)
  – Then \( \text{entity}_b^K \in Part_A \) may be induced
  – Decision to induce membership is an attribute of the mesh part

• Example:
  – Entity \((3,9) \in \text{Part “Block-2”}\)
  – Nodes and face are induced members of Part “Block-2”
Field Data ⊂ Computation Data

• A Computational Mesh Supports Computations
  – Computations require field data:
  – Data associated with mesh entities: \( \text{entity}_a^J \rightarrow \{ \text{FieldData}_X \} \)
  – Heterogeneity: existence of field data varies with mesh entity

• Field Declaration \( \text{Field}_X \)
  – The type of the field; analogous to a ‘C’ or ‘C++’ typedef
  – A multidimensional array of a simple mathematical type

• Field Restriction \( (\text{Field}_X, J, \text{Part}_A) \rightarrow [n_0, n_1, \ldots] \)
  – A field exists for an \( \text{entity}_a^J \)
    • Which is of a specified rank \( J \) and
    • A member of a specified mesh part \( \text{entity}_a^J \in \text{Part}_A \)
  – Defines the dimensions of the multidimensional array value
Field Data Heterogeneity Example

• Velocity field data on all nodes
  – Velocity field declaration
  – Restriction \((\text{velocity}, 0, \text{Part}_\Omega) \rightarrow [3]\)

• Pressure field data only on vertices
  – Pressure field declaration
  – Vertex node mesh part \(\text{Part}_{\text{VTX}}\)
  – Restriction \((\text{pressure}, 0, \text{Part}_{\text{VTX}}) \rightarrow 1\)
  – Vertex nodes declared to be members of vertex node mesh part
Heterogeneity Impacts Performance

• Heterogeneous computations, heterogeneous field data
  – Computations operate on subsets of $\Omega$
    • Problem defined: e.g., fluid region, solid region, boundary
    • Discretization defined: e.g., hex, tet, linear, quadratic, shell
    • Parallel defined: e.g., owned by local process
  – Field data existence and dimensions can vary across $\Omega$

• Impact on Performance
  – Want: Computations on nice contiguous arrays of field data
  – Have: Irregular computations and irregular field data
    • Selection logic in inner loops hurts performance, esp. GPGPU
    • Dense arrays with “ignore this entry” flags wastes memory
  – Solution: …
Field Data Arrays in **Buckets**

• We have **homogeneous subsets** of mesh entities
  – Same mesh entity rank and members of same mesh parts
  ⇒ Have same field data of the same array dimensions

• **Buckets** of homogeneous field data
  – Contiguous arrays of field data
  – Bundled into a block of memory

• **Computations on buckets**
  – Outer loop to select buckets
  – Inner loop to computes on arrays in the selected bucket
  – Active R&D for **portable thread-parallelism**
    • Including GPGPU – buckets residing in device memory
Constructing Mesh Meta-Data  
(mesh database schema)

• Declare Mesh Parts and Mesh Fields
  – And mesh part superset-subset relationships \( Part_A \subseteq Part_B \)
  – And mesh field restrictions \( (Field_X, J, Part_A) \rightarrow [n_0, n_1, \cdots] \)
  – Detect and prohibit inconsistencies
    • Cyclic superset-subset relationships
    • Conflicting field array dimensions

• Complete (Finalize) Mesh Meta-Data Construction
  – Analogous to a database schema for the mesh bulk-data
  – Prohibit changes after mesh bulk-data is created …
  – Because it can be expensive and complex to edit a populated database’s schema
Modifying Mesh Bulk-Data
(structural changes)

• Two Mesh Bulk-Data Modification States

  Parallel Consistent → restore parallel consistency → Modifiable (structurally)

• Structural modifications
  – Declare and destroy mesh entities and relations
  – Change mesh entities’ mesh part memberships
  – Local and parallel-inconsistent for shared or ghosted entities

• Restore parallel consistency
  – A single parallel collective operation
  – Very complex; good performance is especially hard
  – Incremental – only resolve what has been modified
STK-Mesh Finite Element Layer

• Layer element concept onto “generic” mesh
  – Element shapes and node ordering (a.k.a., cell topology)
    • Including element-sides and element-edges
  – Trilinos / Shards API and library of standard cell-topologies

• Optionally Associate a Cell Topology with a Mesh Part
  – \( Part_{Tet}(4) \rightarrow Tetrahedron(4) \)
  – All elements that are members of this part are tetrahedrons

• Includes concept of element boundaries
  – Sides & side neighbors, edges & edge neighbors

• Foundation for finite element computations
  – Basis functions, numerical integration, …
Conclusion

• STK-Mesh is an active R&D effort
  – Within the DOE ASC SIERRA Toolkit project at Sandia
  – Related R&D for field data arrays on multicore and GPGPU

• Very complex domain and needs
  – Parallel, heterogeneous, dynamically modifiable unstructured mesh
  – Computational performance requirements: field data buckets

• Domain Modeling is Key to Managing Complexity
  – Modular and layered architecture
  – “Lean and clean” modules, dependencies, and APIs
  – Minimize coupling between modules