FieldML – a meta-language for field interchange

Richard Christie, Poul Nielsen, Chris Bradley, Caton Little, Randall Britten, Peter Hunter

Auckland Bioengineering Institute

CellML workshop, Auckland, 11 April 2011

AUCKLAND BIOENGINEERING INSTITUTE
THE UNIVERSITY OF AUCKLAND NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

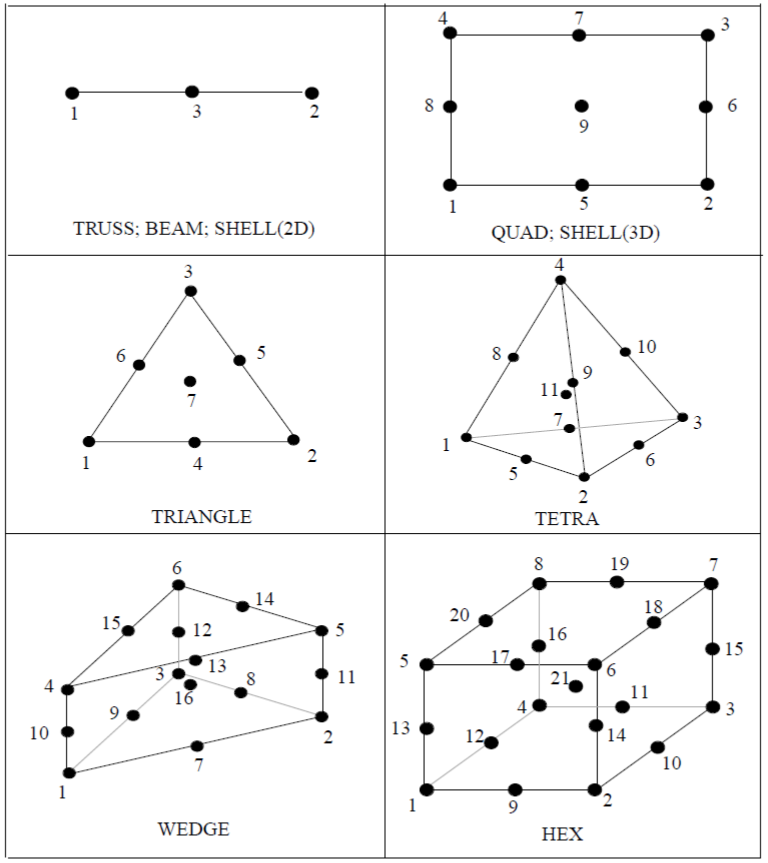
Requirements for FieldML

- A standard format for interchanging field descriptions and data between different software.
- Able to describe fields of arbitrary complexity.
- Efficient.
- Extensible.
- Reusable model components.

Existing Field Formats

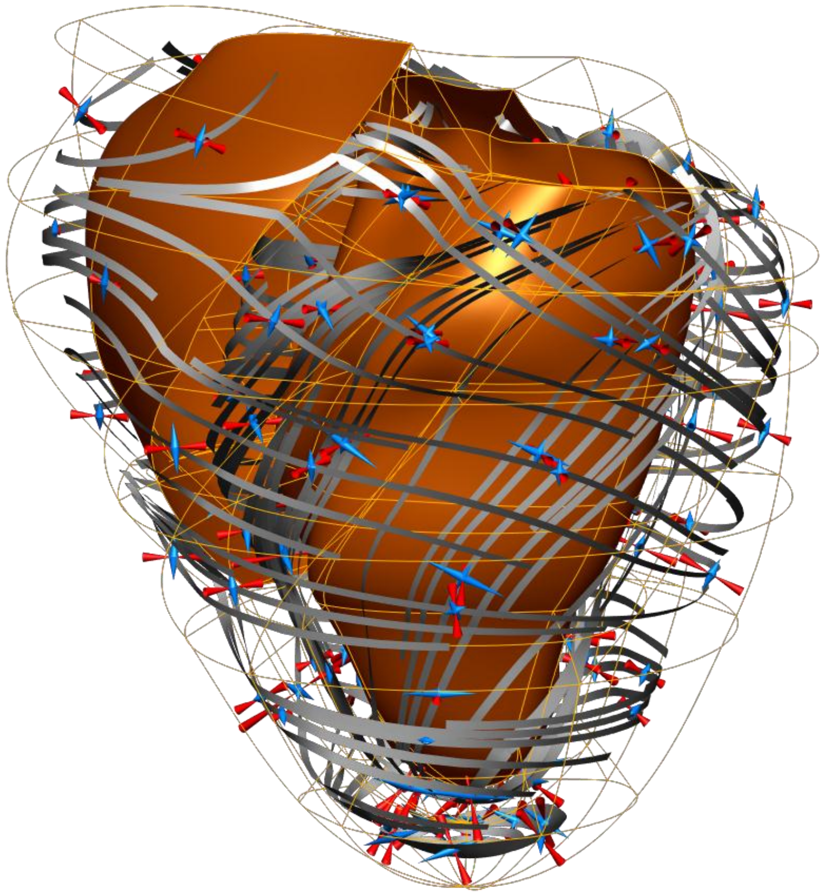
Typical features:

- Fixed ‘element types’ combine shape and function
- ‘Nodal’ interpolation
- Special treatment of coordinates
- Unstructured or limited structured mesh support
- Large number of concepts
- Use standard field names to convey meaning



Exodus II format, Sandia National Labs

Background: CMISS/cmgui



- 'Everything is a field': geometry, material properties, solution, derived values.
- Separation of shape and function.
- High order basis functions and complex parameter mappings.
- Time indexed parameters.
- Fields derived by mathematical expressions on other fields incl. composition/embedding.
- Image & image processing fields.

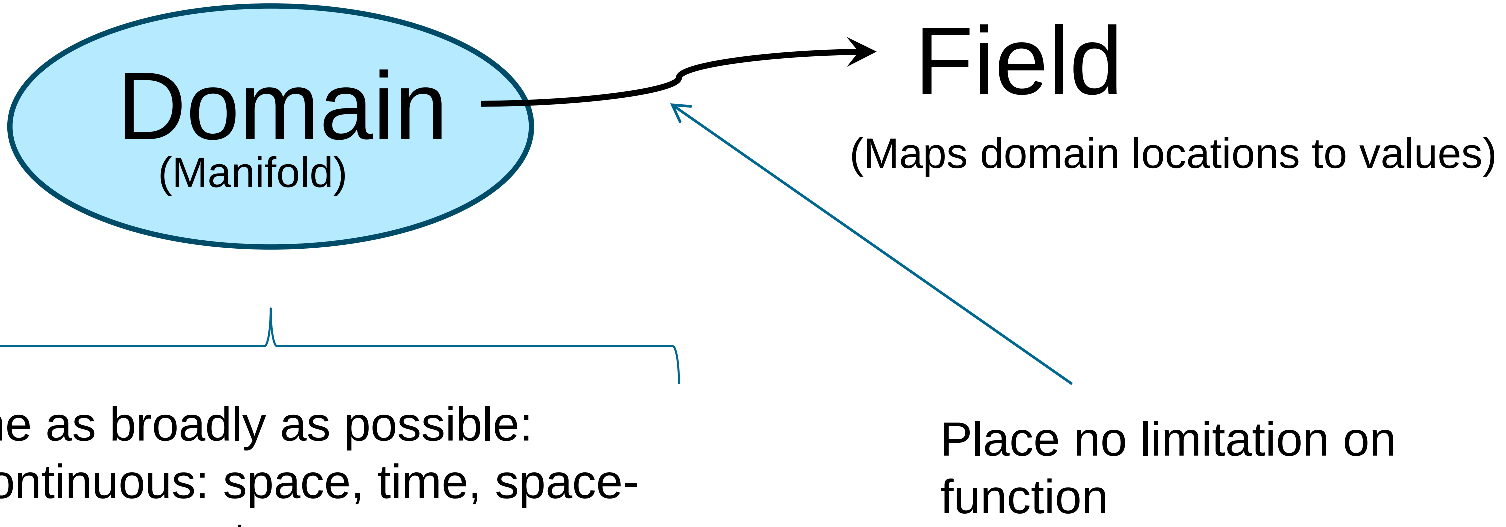
BUT:

- Fixed function interpolation.
- Only linear time variation.
- Object overheads.

FieldML Design Philosophy

- A meta-language for applications to describe their structures and data in a standard way.
- Base on minimal set of fundamental concepts.
- Avoid fixed, legacy structures esp. low-level objects, their limitations and overheads.
- Simple if simple, complex if complex.
- Preference for homogeneous data.
- Separation of bulk data from high-level description.
- Use metadata to communicate purpose of fields and specialised relationships.

Domains and Fields


Domain
(Manifold)
Field
(Maps domain locations to values)
Place no limitation on function

Define as broadly as possible:

- Continuous: space, time, space-time, parameter-space…
- Discrete: points, nodes, population…

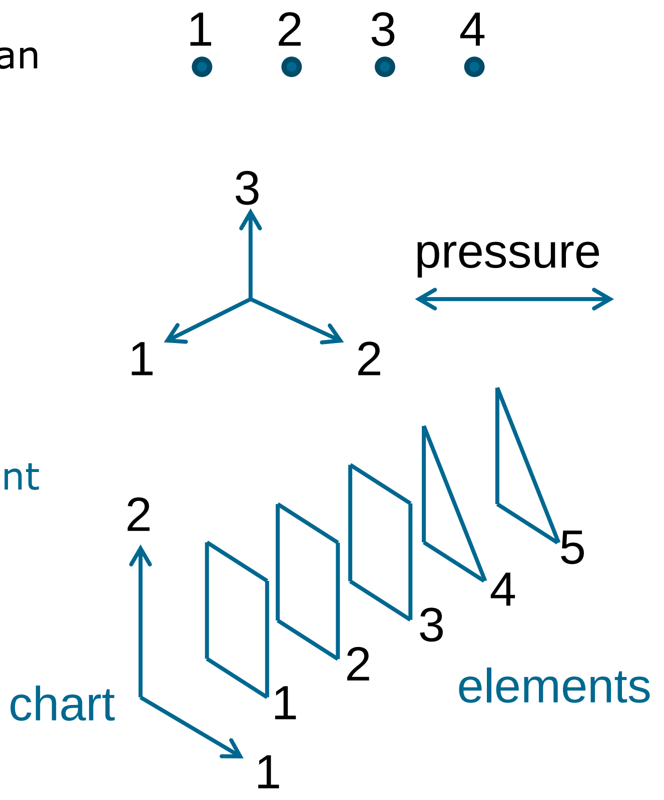
But maintain manifold of fixed dimension.

… so all data is ‘field-like’

FieldML Concepts: Types

Reusable descriptions of spaces (manifolds) from which field domains are built.

- *Discrete 'ensemble' type:* set of unique identifiers, an enumeration: nodes, elements, components, time indexes, etc.
- *Continuous type:* N-dimensional chart indexed by component ensemble type, possibly bounded: coordinate system ($\mathbb{R}^N$), scalar domains e.g. temperature, pressure ($\mathbb{R}^1$), tensor domains, etc.
- *Structured type:* tuple of other types with arbitrary bounds: mesh (a tuple of discrete elements and continuous element chart with bounds of the element chart given as a function of element and chart position).



FieldML Concepts: Evaluators

- Reusable objects for building field evaluation pipelines.
- FieldML domains have no attributes except those mapped by evaluators.

Evaluators:

- … encapsulate an operator / function,
- … acting on values of 0 or more ‘input’ evaluators,
- … to produce values of a prescribed type.

6 sub-classes of evaluators:

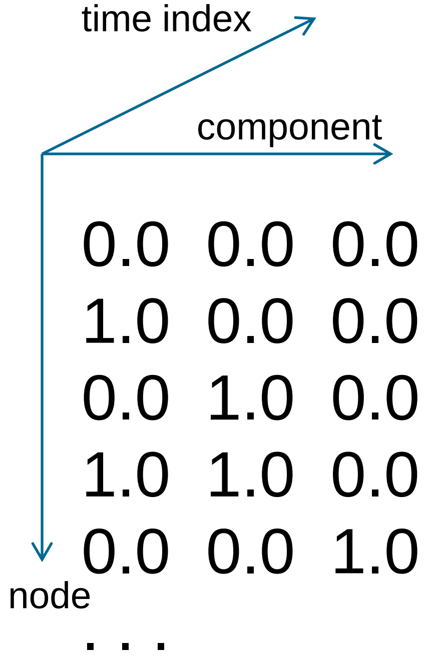
Abstract, Parameters, Reference, Piecewise, Aggregate and External.

Abstract Evaluator

- The initial inputs to evaluation pipelines ~ function arguments.
- Represents a value source of prescribed type.
- Targets for binding: all abstracts must be bound to a concrete evaluator or given literal values to evaluate pipeline.
- FieldML 0.3: unbound abstract evaluators can be considered parts of field domain atlas.

Parameters Evaluator

- Source of literal data in FieldML data model.
- Continuous or ensemble value.
- Parameters indexed by N (≥ 0) ensemble-valued input evaluators.
- Parameter values in external resources or inline in XML.
- Equivalent to a multi-dimensional array, dense or sparse.
- Important: can substitute any other evaluator for fixed parameters.



Reference Evaluator

- Evaluates a referenced evaluator, but:
- Generally binds new evaluators to one or more abstract sources of referenced evaluator to modify its behaviour.
- Similar to a function call mechanism, but arguments are functions rather than simple values.
- Only object able to reference evaluators from outside the current scope in the FieldML document.

Piecewise Evaluator

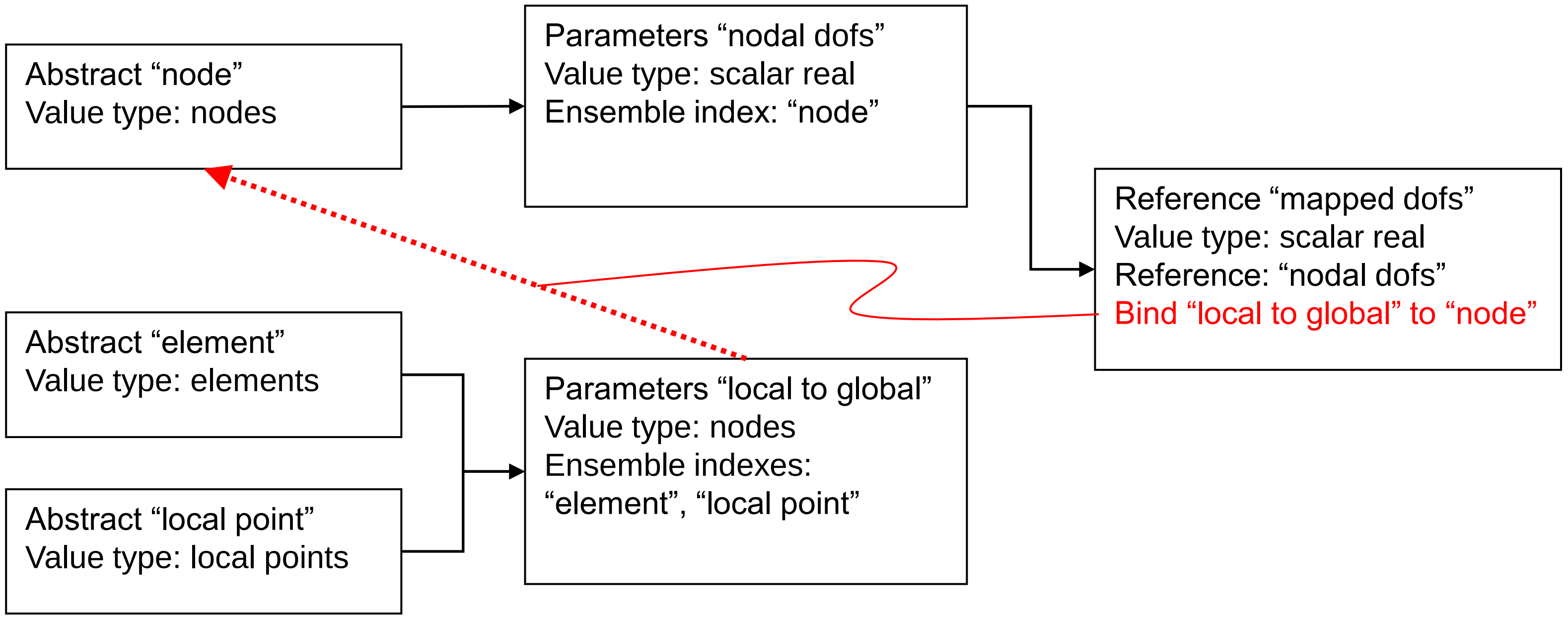
- ‘Switch’ operator delegating which evaluator to use, depending on value from single ensemble-valued input evaluator.
- Often used to implement fields defined over meshes with different basis functions in each element.

Aggregate Evaluator

- Aggregates referenced scalar evaluators over an ensemble to produce a vector valued result.
- Future: support aggregation of multiple indexes for tensor value types, aggregation of structured types.

External Evaluator

- Named standard evaluators of agreed functionality.
- Primary extension mechanism in FieldML, usually only seen in FieldML / third-party library serialisations.
- Implementation e.g. MathML may be present, but operator may alternatively be ‘understood’ by name.
- Initial FieldML library supports standard element interpolation and shape functions as external evaluators. Future: support any required mathematical or other operators.
- Applications parsing FieldML must map all evaluators – particularly external evaluators – to fixed objects in their data model which perform the equivalent functionality (if supported).



Evaluator Binding

Use in mapping 'nodal' parameters to local element.

Note: binding is deeply functional, not value passing.

FieldML Features in Development

- Regions, for building models out of sub models.
- Formal declarations of Domains and Fields
  c.f. interpreting evaluators as fields.
- Domains with multiple atlases, fields defined on different atlases of same domain.
- Embedding.
- Ensemble subsets and sequences for arbitrary parameter ordering.
- Hierarchical domains.
- Physical units.
- Efficient binary bulk data stores, e.g. HDF5.

FieldML benefits for large, multi-domain, multi-scale, multi-variate problems.

- Uniform treatment of components of field domains: same basis functions + mappings usable for space, time, patient weight, population…
- Uniform treatment of structured and unstructured meshes: structured meshes differ only in using a formulaic parameter map; efficient ‘semi-structured’ meshes e.g. all elements of same shape.
- Uniform treatment of formerly disparate data types: Finite element interpolated variables, images, CAD geometry are all fields.
- Support for embedding, hierarchical models.

FieldML Resources

- Website: http://www.fieldml.org/
  Look out for upcoming document:
  "FieldML 0.3 Concepts and Serialisation"
- FieldML API, prototypes and mock-ups:
  http://code.google.com/p/fieldml/
- Model repository: http://models.fieldml.org/
- Developer e-mail:
  fieldml-developers@lists.sourceforge.net
- Tracker: https://tracker.physiomeproject.org/

Details are subject to change, particularly up to end of May 2011.
We invite collaboration and feedback.

AUCKLAND BIOENGINEERING INSTITUTE
THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

Acknowledgements

- FieldML has been developed with assistance from the following organisations:

Virtual Physiological Human
network of excellence

euHeart

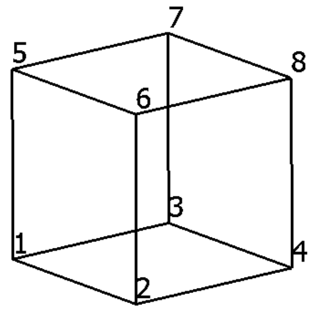
wellcometrust

MAURICE WILKINS CENTRE
FOR MOLECULAR BIODISCOVERY

Specific grants:

- 9077 / 3622407 European Commission – FP7 VPH NoE
  Virtual Physiological Human Network of Excellence
- 9077 / 3625656 European Commission – FP7 International MSV
  Multi-scale Spatio-temporal Visualisation

Example: cube.xml



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fieldml version="0.3_alpha" xsi:noNamespaceSchemaLocation="Fieldml_0.3.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<Region name="cube" library="library_0.3.xml">

<!-- define ensemble/set of 8 entries to represent nodes 1..8 -->
<EnsembleType name="cube.nodes">
      <bounds>
            <ContiguousEnsembleBounds valueCount="8"/>
      </bounds>
</EnsembleType>

<AbstractEvaluator name="cube.nodes.variable" valueType="cube.nodes"/>

<!-- declare a variable placeholder for parameters to a field template, in this case anticipated to vary with ensemble "cube.nodes" -->
<AbstractEvaluator name="cube.dofs_variable.nodal" valueType="library.real.1d"/>

<!-- define a 3-D mesh consisting of one element of unit cube shape. Mesh dimensionality of 3 is indicated by 3-component library ensemble
"library.ensemble.xi.3d" specified for xi_component attribute -->
<MeshType name="cube.mesh" xi_component="library.ensemble.xi.3d">
      <bounds>
            <ContiguousEnsembleBounds valueCount="1"/>
      </bounds>
      <shapes default="library.shape.cube"/>
</MeshType>

<AbstractEvaluator name="cube.mesh.variable" valueType="cube.mesh" />
```

```
 <!-- define mapping from element*local-node to global index from ensemble "cube.nodes". Local-node arrangement "library.local_nodes.cube.2x2x2" is
documenting in the library as being the 8 corner points of a unit cube at xi = (0,0,0), (1,0,0), (0,1,0), (1,1,0), (0,0,1), (1,0,1), (0,1,1), (1,1,1) -->
<ParametersEvaluator name="cube.layout_2x2x2" valueType="cube.nodes">
    <SemidenseData>
        <denseIndexes>
            <index evaluator="library.local_nodes.cube.2x2x2.variable"/>
            <index evaluator="cube.mesh.variable.element"/>
        </denseIndexes>
        <dataLocation>
            <inlineData>
                1 2 3 4 5 6 7 8
            </inlineData>
        </dataLocation>
    </SemidenseData>
</ParametersEvaluator>

<!-- construct a vector of nodal parameters to pass on to "cube.trilinear.interpolator" -->
<AggregateEvaluator name="cube.trilinear.parameters"
    valueType="library.parameters.trilinear_lagrange">
    <binds>
        <bind_index variable="library.local_nodes.cube.2x2x2.variable"
            index_number="1"/>
        <bind variable="cube.nodes.variable" source="cube.layout_2x2x2"/>
    </binds>
    <componentEvaluators default="cube.dofs_variable.nodal" />
</AggregateEvaluator>
```

cube.xml 2

```
 <!-- parameters for the coordinate field, listing a scalar real parameter for all permutations of 3-entry
library component ensemble "library.ensemble.rc.3d" and 8-entry ensemble "cube.nodes" -->
<ParametersEvaluator name="cube.corner_coordinates" valueType="library.real.1d">
	<SemidenseData>
		<denseIndexes>
			<index evaluator="library.ensemble.rc.3d.variable"/>
			<index evaluator="cube.nodes.variable"/>
		</denseIndexes>
		<dataLocation>
			<inlineData>
				0.0 0.0 0.0
				1.0 0.0 0.0
				0.0 1.0 0.0
				1.0 1.0 0.0
				0.0 0.0 1.0
				1.0 0.0 1.0
				0.0 1.0 1.0
				1.0 1.0 1.0
			</inlineData>
		</dataLocation>
	</SemidenseData>
</ParametersEvaluator>

<!-- define evaluator returning value of library FEM basis evaluator "library.fem.trilinear_lagrange" at the element chart location of mesh type
"cube.mesh" (denoted by "cube.mesh.xi") and using parameters from evaluator "cube.trilinear.parameters". -->
<ReferenceEvaluator name="cube.trilinear.interpolator"
  evaluator="library.fem.trilinear_lagrange"
  valueType="library.real.1d">
	<binds>
		<bind variable="library.xi.3d.variable" source="cube.mesh.variable.xi"/>
		<bind variable="library.parameters.trilinear_lagrange.variable"
			source="cube.trilinear.parameters"/>
	</binds>
</ReferenceEvaluator>
```

cube.xml 3

```
<!-- define a piecewise template delegating which evaluator gives the template its values in each element, which is trivial for this one element mesh. It is
a template for a field defined over the mesh represented by "cube.mesh.variable", with the unbound parameter source "cube.dofs_variable.nodal"
inherited from delegate evaluator "cube.trilinear.interpolator" -->
<PiecewiseEvaluator name="cube.template.trilinear" valueType="library.real.1d">
    <binds>
        <bind_index variable="cube.mesh.variable.element" index_number="1" />
    </binds>
    <elementEvaluators>
        <element number="1" evaluator="cube.trilinear.interpolator"/>
    </elementEvaluators>
</PiecewiseEvaluator>

<!-- define the final vector [coordinate] field by aggregating evaluators for each component of the vector valueType. Although each component uses the
same evaluator in this example, they produce different values because the parameters on which they depend vary with the same component ensemble
("library.ensemble.rc.3d") which the library documents as indexing this field's valueType (continuous type "library.coordinates.rc.3d"). Hence only one
binding is required to set the parameter source for this field -->
<AggregateEvaluator name="cube.coordinates" valueType="library.coordinates.rc.3d">
    <binds>
        <bind_index variable="library.ensemble.rc.3d.variable" index_number="1" />
        <bind variable="cube.dofs_variable.nodal" source="cube.corner_coordinates"/>
    </binds>
    <componentEvaluators>
        <component number="1" evaluator="cube.template.trilinear"/>
        <component number="2" evaluator="cube.template.trilinear"/>
        <component number="3" evaluator="cube.template.trilinear"/>
    </componentEvaluators>
</AggregateEvaluator>

</Region>
</fieldml>
```

cube.xml 4