Modeling as a Service
NGMF (OMS3), Terracotta, and EC2
NGMF – A NONINVASIVE MODELING FRAMEWORK
Env. Modeling Frameworks

• Support the development & application of environmental models
  – Coupling, Execution management
  – Analysis, Statistics, Visualization
  – Spatial / Temporal relationships and management
  – Calibration/Parameter Estimation/Uncertainty Analysis
  – Data I/O
Env. Modeling Frameworks

• They are optional!
• Driven by research organizations
• Reflecting a domain views
• Goals
  – Eliminate redundancy
  – Improve efficiency
  – Simplify complicated tasks
Major Efforts

- Earth System Modeling Framework
- Common Component Architecture
- Object Modeling System
- OpenMI
- Frames
- ...
Issues

• Infrastructure complexity
• Commitment level
• Learning curve
• App and Framework lifelines
• Legacy user code
• API/Type overlap

• Invasive vs. non-invasive
Framework Invasiveness

- How does the use of a particular framework impact an application's quality?
- Quality with respect to
  - Portability
  - Reusability
  - Understandability
  - Maintainability
Framework Invasiveness - 2

- Coupling between application and framework code
  - Framework APIs
  - Framework Interfaces
  - Framework Custom Data Types

- Mass of boilerplate code
  - Mass should account for both LOC and source code size in bytes

- External framework dependencies
  - Required non-framework APIs
Framework Invasiveness – 3

- Other framework couplings
  - Language dependencies
  - Platform dependencies
    - For example CCA required Redhat 8

Organizational dependency
- Training cost/investment
- Hardware cost/investment
- Support cost/investment
Research Approach

- How does framework invasiveness impact software quality?
- How do we measure framework to application invasiveness?

Analysis of models implemented in various frameworks, languages
Invasiveness Properties

- Number of Framework specific data types used/uses
- Number of framework specific functions used/uses (API calls)
- LOC of model implementation
  - Boilerplate LOC of model implementation
  - Application LOC of model implementation
- Boilerplate to Application LOC ratio
- Size of Framework
  - Number of classes/modules/functions
  - Size (LOC)
Thornthwaite Model

- Water balance model
  Models distribution of water among components of hydrological system
  (Thornthwaite, 1948; Mather, 1978; 1979)

- Implemented in
  4 Languages
  5 Frameworks

- How do implementations differ?
ThornthwaiteMC Simulation Model

Random number generation in @Range

Monthly Input
- Climate
  - Precipitation
  - Temperature
  - Month

Monthly Output
- Surface Runoff
- Soil Moisture
- Actual ET
- Snow Storage
- Potential ET

Parameter
- Latitude
- Soil Moisture Capacity
- Runoff Factor

Monthly Input
- DayLen
- HamonET
- Soil Moisture
- Snow
- Runoff

Thornthwaite Monte Carlo Simulation

Olaf David
Source Code Size

- Thornthwaite

ESMF C implementation uses global data

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain C++</td>
<td>12437</td>
</tr>
<tr>
<td>OMS Java</td>
<td>21846</td>
</tr>
<tr>
<td>NGMF Java</td>
<td>11843</td>
</tr>
<tr>
<td>ESMF C</td>
<td>24002</td>
</tr>
<tr>
<td>ESMF Fortran</td>
<td>44473</td>
</tr>
<tr>
<td>Plain Fortran</td>
<td>10678</td>
</tr>
<tr>
<td>CCA Java (java source only, no infrastructure)</td>
<td>581901</td>
</tr>
<tr>
<td>OpenMI Java</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Methods

• Use Annotations/metadata over API
• Minimize the API or eliminate it entirely.

• Implementation Techniques
  – Annotations / Attributes
  – Runtime introspection / Reflection
  – Dynamic class loading
NGMF

• Next Generation Modeling Framework (work title)
  – Component-based
  – lightweight (non-invasive)
  – Dataflow driven execution
  – Default Multithreaded execution
NGMF & OMS

• What is common?
  – Conceptually similar, Init/Run/Cleanup

• What is different?
  – No Framework data types in NGMF
  – No interfaces provided by NGMF
  – Default Multithreading in NGMF
  – NGMF Annotated POJOS vs. OMS2 API Interface implementations
A Component

input(s) -> processing -> output(s)

Olaf David
Example Component

\[ area = \pi \times \text{radius}^2 \]

- **Input**: radius
- **Output**: area

\( \bullet \) Input output
import ngmf.ann.*;

@Description("Circle computation")
@Author("me")
public class CircleArea {

    @Description("Radius")
    @Range(min=0)
    @In public double r;

    @Out public double area;

    @Execute
    public void runme(){
        area = Math.PI * r * r;
    }
}

Olaf David
Simple Example component. **Any Plain Old java Object (POJO) is a component in NGMF.**

```java
class Component {
    ...
}
```

- Public component class
- Default public, non argument constructor. It can be omitted if there is no other constructor given.
- No need to subclass or implement interfaces.

Component

= POJO + Meta data
Meta data purpose

• Execution control and connectivity
• Execution support
• Documentation/Repository support
• Testing support
• Runtime consistency support
<table>
<thead>
<tr>
<th>Component</th>
<th>Field</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Description</td>
<td>@Description</td>
<td>@Execute</td>
</tr>
<tr>
<td>@Author</td>
<td>@Unit</td>
<td>@Initialize</td>
</tr>
<tr>
<td>@Bibliography</td>
<td>@In</td>
<td>@Finalize</td>
</tr>
<tr>
<td>@Status</td>
<td>@Out</td>
<td></td>
</tr>
<tr>
<td>@VersionInfo</td>
<td>@Range</td>
<td></td>
</tr>
<tr>
<td>@SourceInfo</td>
<td>@Role</td>
<td></td>
</tr>
<tr>
<td>@Keywords</td>
<td>@Bound</td>
<td></td>
</tr>
<tr>
<td>@Label</td>
<td>@Label</td>
<td></td>
</tr>
</tbody>
</table>
Description

A component with more descriptive information.

```java
@Description("Circle Area Calculation.")
public class CircleArea {
    @Description("Radius")
    @In public double r;
    ...
}
```

- Tag the Component and the fields being used for input and output with the `@Description` annotation.
- This is optional metadata.
Simple Example component with author information.

```java
@Author(
    name="Joe Scientist",
    org="Joe’s Research Org",
    contact="joe.scientist@research-org.edu")

public class ET {
}
```

- `@Author` provides some contact information.
- Attribute name is mandatory, `org` and `contact` are optional
Example component with Bibliography information.

```java
@Description("Circle Area Calculation.")
@Bibliography("Journal of Geometry, Vol.1, p..")
public class CircleArea {

    ...

}
```

- Attach a `@Bibliography` annotation to a component to refer to Literature background, Web sites that contain detailed documentation, etc.
- This annotation does have the same purpose like a Bibliography list in a scientific paper.
- Each reference is a separate string, multiple references are comma separated.
- This is optional metadata and can be used to document components.
Component with status information. A status is a component quality indicator.

```java
@Description("Circle Area Calculation.")
@Status(Status.TESTED)
public class CircleArea {
    ...
}
```

- Component have an optional status.
- A developer can specify the level of completeness or maturity of a component with this tag.
- Predefined values
  - DRAFT, SHARED, TESTED, VALIDATED, CERTIFIED
- This is optional metadata and can be used to classify, verify stored components in a repository
Simple Example component with version information. Can (Should) be used in conjunction with a VCS.

```java
@VersionInfo("$Id: ET.java 20 2008-07-25 22:31:07Z od $")
public class ET {
}
```

- @VersionInfo might contain more than just a number.
- Version control systems such as CVS, Subversion, or Mercurial provide keyword substitution that present revision number, last modification time, or developer id.
- @VersionInfo is optional but is good practice.
- Component repositories can use and present this information

http://svnbook.red-bean.com/en/1.5/svn.advanced.props.special.keywords.html
http://www.selenic.com/mercurial/wiki/index.cgi/KeywordPlan
Simple Example component with source information. Can (Should) be used in conjunction with a VCS.

```java
@SourceInfo("$HeadURL: http://www.test.org/repo/ET.java $")
public class ET {
}
```

- `@SourceInfo` provides some link to the source.
- Version control systems such as CVS, Subversion, or Mercurial provide keyword substitution that fills in the Repository URL.
- `@SourceInfo` is optional.
- Component repositories can use and present this information

http://svnbook.red-bean.com/en/1.5/svn.advanced.props.special.keywords.html
http://www.selenic.com/mercurial/wiki/index.cgi/KeywordPlan
Example component with descriptive information.

@Description("Circle Area Calculation.")
@Keywords("Geometry, 2D")
public class CircleArea {
    ...
}

- Tag the Component with the @Keywords annotation to characterize it.
- This annotation does have the same purpose like a Keyword list in a scientific paper.
- This is optional metadata and can be used to index, search, and retrieve archived and stored components. It is optional meta data.
Simple Example component with label information. Labels relate to ontologies (label is an OWL annotation)

```java
public class Calc {
    @Label("latitude")
    @In public double lat;
    ...
}
```

- Labeling a field or component offers alternative names.
- Labels might be used to relate components or fields to ontologies.
- Labels are optional.

http://www.w3.org/TR/owl-ref/
Simple Example component with execution method. This method provides the implementation logic of the component where the input is being transformed to output.

```java
public class Component {
    @Execute
    public void executemethod() {
        // execute code here
    }
}
```

- Name the execution method any name you want, but annotate it with `@Execute`
- The execute methods has to be non-static, public, void, no arguments.
- This is required meta data
Simple Example component with init method. In this method the internal state of a component should be initialized. (e.g. opening a file for reading)

```java
import ngmf.ann.*;
public class Component {
    @Initialize
    public void start() {
        // initialization code
    }
}
```

- Name the execution method any name you want, but annotate it with `@Initialize`
- The init methods has to be non-static, public, void, and has no arguments.
- This method gets called once after Component instantiation and before the first execution.
- This is optional meta data
Simple Example component with finalize method. This method provides the notion of a final cleanup after model execution (e.g. closing a DB connection)

```java
public class Component {
    @Finalize
    public void cleanup() {
        // execute code here
    }
}
```

- Name the finalization method any name you want, but annotate it with `@Finalize`
- The method has to be non-static, public, void, and has no arguments.
- Finalize overlaps with Java’s `finalize()` method that gets called from the garbage collector.
- This is optional meta data
In & Out

Simple Example component with annotated I/O fields.

```java
public class CircleArea {
    @In  public double radius;
    @Out public double area;

    @Execute
    public void runme() {
        area = Math.PI * radius * radius;
    }
}
```

- Tag the fields being used for input and output with `@In` and `@Out`.
- Fields can have any data type (primitive or objects) and should be public.
- No arguments for `@In` and `@Out`.
- Required Meta data.
Simple Example component with unit information.

```java
public class Calc {
    @Unit("degree")
    @In public double latitude;
    ...
}
```

- Unit information for IO fields
- Usually used for scalars and arrays.
- Allows frameworks to support unit checking/validation and conversion
- Optional Meta data.

http://jscience.org
Ranges

Simple Example component with range information.

```java
public class Calc {
    @Unit("degree")
    @Range(min = -90, max = 90)
    @In public double latitude;

    ...
}
```

- Range information for valid input and output values.
- Min and max inclusive.
- Default value for min if omitted: Double.MinValue
- Default value for max if omitted: Double.MaxValue
Ranges (cont.)

- Uses cases for Range
  - `@Range(min=0) // value has to be >=0`
  - `@Range(min=32, max=50)`

- Use cases
  - Runtime value exchange check
  - Component testing based on statistical data
Field with role information. A role is a modeling concept such as a ‘parameter’ or ‘state variable’ that applies for a field.

```java
public class ET {
    @Description("Jensen-Haise Coefficient")
    @Role(Role.Parameter)
    @In public double jh_coeff;
    ...
}
```

- Tag the Field with the `@Role` annotation to characterize its role in this component.
- Choose from a predefined set: `Role.Parameter`, `Role.Variable`, `Role.Simulated`, `Role.Observed`) or create your own `Role`
- A Role annotation takes a string argument.
- This is optional meta data.
A field might have bound information. A bound defines a binding to another field. This could be a dimension for an array.

```java
public class ET {
    @Bound("nsim")
    @In public double[] jh_coeff;
    @In public int nsim;
    ...
}
```

- A Bound annotation takes a string argument.
- It allows a GUI to present dependencies between fields.
- This is optional meta data.

- Jh_coeff has the named dimension nsim.
Component Metadata Variants

• 3 different syntaxes to provide for the same Component meta data semantics using (i) Annotations or (ii) XML

1) Embedded Metadata using Annotations
2) Attached Metadata using Annotations
3) Attached Metadata using XML
1) Embedded Metadata Annotations

```java
import ngmf.ann.*; // (1)

public class CircleArea {
    @In public double radius; // (2)
    @Out public double area; // (2)

    @Execute
    public void runme() {
        area = Math.PI * radius * radius; // (3)
    }
}
```

CircleArea.java

- The metadata is part of the component via Annotations
- Must have access to source code
2) Attached Metadata Annotations

Two Separate files!

```java
public class CircleArea {
    public double radius;
    public double area;

    public void exec() {
        area = Math.PI * radius * radius;
    }
}
```

**CircleArea.java**

```java
import ngmf.ann.*; // (1)

public abstract class CircleAreaCompInfo {
    @In public double radius; // (2)
    @Out public double area; // (2)

    @Execute
    public abstract void exec();
}
```

**CircleAreaCompInfo.java**

- POJO, no NGMF dependency
- class/jar sufficient, no source required
- Abstract Class (cannot run), abstract methods
- Required ‘CompInfo’ Extension
- Names for Component, fields and methods must match
3) Attached Metadata XML

```java
public class CircleArea {
    public double radius;
    public double area;

    public void exec() {
        area = Math.PI * radius * radius;
    }
}
```

```xml
<CircleArea:Component>
    <radius:In/>
    <area:Out/>
    <exec:Execute/>
</CircleArea:Component>
```

**CircleArea.java**

- POJO, no NGMF dependency
- class/jar sufficient, no source required

**CircleAreaCompInfo.xml**

- XML namespaces for component and field name
- Annotations → XML
3) Attached Metadata XML (cont)

@Description(“Groundwater Reservoirs”)
@Range(min=10, max=20)
@In public int ngwres;
...

<gwres:Description>
  Groundwater Reservoirs
</gwres:Description>
<gwres:Range min='10' max='20'/>
<gwres:In/>

• Java Annotation – XML equivalence
  – Annotation Type name ↔ QName Local part
  – Field name ↔ QName Prefix
  – Annotation attribute ↔ Element Attribute
  – Annotation attribute value ↔ Element Attribute value
  – Annotation content ↔ Element PCDATA content

http://www.w3.org/TR/REC-xml/
NGMF COMPONENT INTEGRATION
Compound

• Is a more complex component (but still POJO)
• Contains other, internal components
• Responsible for internal connectivity, In/Out mapping
• Contains an internal NGMF Controller
Cylinder Compound

CylinderCompound

height

mapIn

rad

mapIn

CircleArea

radius

area

connect

CylinderSurface

height

surface

mapOut

CirclePerimeter

radius

perimeter

connect

Cylinder Compound

Olaf David
public class CylinderCompound extends Compound {
    @In  public double rad;
    @In  public double height;
    @Out public double surface;

    Object p = new CirclePerimeter();
    Object s = new CylSurface();
    Object a = new CircleArea();

    public CylinderCompound () {
        mapIn("height", s, "height");
        mapIn("rad", a, "radius");
        mapIn("rad", p, "radius");

        connect(a, "area", s, "area");
        connect(p, "perimeter", s, "perimeter");

        mapOut("surface", s, "surface");
    }
}
connect(..)

• Connects an @Out with an @In field of two components

• Syntax

  connect(Object out, String outfield,
           Object in, String infield);

  connect (Object out, String field,
           Object...in);

  (if the same field name is being used in out and in objects)
mapIn()

• Maps an Compound @In with an @In field of an inner component

• Syntax

  mapIn(String infield, 
           Object in, String infield1);

  mapIn(String field, Object...in); 
  (if the same field name is being used in all ‘in’ objects)
mapOut()

- Maps an Compound @Out with an @Out field of an inner component
- Syntax

  ```java
  mapOut(String outfield, 
           Object out, String outfield1);
  mapOut(String field, Object... out);  
  (if the same field name is being used in all ‘out’ objects)```
Example: Geoprocessing

Diagram showing the process of geoprocessing with various steps and inputs:

- User
- DEM
- Condition
- Consequence

Steps include:
- Fill
- FlowDir
- FlowAcc
- Cond
- StrmSeg
- ContribAreas

Output: Hillslope

Olaf David
NGMF + EC2 + TERRACOTTA (PREPARATION)
Objective

- Develop a scalable architecture for simulation models based on OMS (NGMF),
- Exercise model deployment into a cloud computing environment such Amazon Web Services
- Prototype an on-demand Service Oriented Architecture using AWS/EC2 for computational intensive models
Terminology

• AWS – Amazon Web Service
  – S3 – Simple Storage Service (part of AWS)
  – EC2 – Elastic Compute Cloud (part of AWS)
  – EBS – Elastic Block Storage (part of AWS)
  – AMI – Amazon Machine Image (Part of EC2)

• TC – Terracotta

• NGMF – Next Generation Modeling Framework (OMS3 core)
Modeling Stack

Model

NGMF

Terracotta

EC2
Scalability Concept

- Single CPU
- Multi Core/CPU
- Cluster (physical/virtual)
Amazon Web Services

- Cloud computing platform provided by Amazon
- Collection of “on-demand” infrastructure services
  - Elastic Compute Cloud (EC2)
  - Simple Storage Service (S3)
  - Simple DB
  - ...
- “Pay as you go”
Terracotta

“Terracotta is open source infrastructure software that makes it inexpensive and easy to scale a Java application to as many computers as needed, without the usual custom application code and databases used to share data in a cluster.

Terracotta manages mission critical data using Network-Attached Memory (NAM) technology.

NAM enables Terracotta to cluster Java Virtual Machines (JVMs) directly underneath applications, and is a proven runtime approach to providing Java applications both high availability and scalability”
General AWS Preparation

• Amazon account
• Sign up for S3 and EC2
• Configured client tools
  – AWS credentials (access key + secret access key)
  – SSH keys
  – Regions (US, EU)
  – Security groups (open ssh and jmx ports)
  – EBS (for installation package)

• Good Clients: AWS (timkay.com), elasticfox, s3fox
Security Groups (instance FW)
Amazon Machine Image Bundling

1. Selected existing AMI (FC8), started
2. Installed Terracotta 2.7, JDK 1.6, NGMF
   - Created EBS install disk (4GB)
3. Bundled/Uploaded/Registered as new AMI
   - 'ngmfimage'
   - "ami-21a74048"
   - private

Olaf David
Published NGMF AMI

### Amazon Machine Images

<table>
<thead>
<tr>
<th>AMI ID</th>
<th>Manifest</th>
<th>Visibility</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>ami-0618fc6f</td>
<td>lingfing-staging/image_bundles/i_7c52815_2008_11_15_12_0</td>
<td>Public</td>
<td>Other Linux</td>
</tr>
<tr>
<td>ami-079d66</td>
<td>yazzgoth-usa/image_bundles/Ubuntu_8_10_Django_SVN_HE</td>
<td>Public</td>
<td>Ubuntu</td>
</tr>
<tr>
<td>ami-21a74048</td>
<td>ngmf/ngmfimage.manifest.xml</td>
<td>Private</td>
<td>Other Linux</td>
</tr>
<tr>
<td>ami-21b35748</td>
<td>tushar-amis-clitest-final-public/image.manifest.xml</td>
<td>Public</td>
<td>Other Linux</td>
</tr>
<tr>
<td>ami-232icb4a</td>
<td>cong-s3-bucket11/image.manifest.xml</td>
<td>Public</td>
<td>Other Linux</td>
</tr>
<tr>
<td>ami-235iba4a</td>
<td>ec2-public-images/getting-started-v1.07.manifest.xml</td>
<td>Public</td>
<td>Other Linux</td>
</tr>
<tr>
<td>ami-29947140</td>
<td>eminent-am/django/10/image.manifest.xml</td>
<td>Public</td>
<td>Other Linux</td>
</tr>
</tbody>
</table>

### 1 Amazon Machine Image selected

<table>
<thead>
<tr>
<th>AMI ID:</th>
<th>ami-21a74048</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner:</td>
<td>136924249431</td>
</tr>
<tr>
<td>Visibility:</td>
<td>Private</td>
</tr>
<tr>
<td>State:</td>
<td>available</td>
</tr>
<tr>
<td>Manifest:</td>
<td>ngmf/ngmfimage.manifest.xml</td>
</tr>
<tr>
<td>Product Code:</td>
<td>-</td>
</tr>
<tr>
<td>Architecture:</td>
<td>i386</td>
</tr>
<tr>
<td>Kernel ID:</td>
<td>aki-a71cf9ce</td>
</tr>
<tr>
<td>Ramdisk ID:</td>
<td>ari-a51cf0cc</td>
</tr>
<tr>
<td>Platform:</td>
<td>Other Linux</td>
</tr>
</tbody>
</table>
Elastic Block Store

4 GB volume, Installation packages (Terracotta, java, ...)

Olaf David
NGMF CLUSTER CONTROL (NC2)
NGMF Cluster Control (nc2)

• Purpose: manage a set of machines
  – Copy files
  – Execute commands
• Physical and Virtual machine cluster
• Part of NGMF library
• Other tools such as C3 (Globus) insufficient
  – No output serialization/merging
  – Insufficient Configuration for EC2, ssh vs. rsync vs. rsh
nc2 Options

- `s`  
  serialize output with node, default is merged output
- `v`  
  verbose output
- `c<cluster name>:<node 0>[<node n>]`  
  select a cluster and node set (default is ‘default’)
  - `c ec2:0-*`  
    all nodes in cluster ‘ec2’
  - `c ec2:0`  
    first node in ‘ec2’
  - `c def:0,1-3,5,7-*`  
    node 0,1,2,3,5,7,rest in ‘def’

- `put <local file> <remote file>`  
  copy <local file> in to <remote file> on nodes in selected cluster
- `exec <command>`  
  executes <command> on nodes in cluster in parallel
Configuration: $HOME/nc2.conf

# location of the ssh executable
ssh = /usr/bin/ssh

# location of the scp executable
scp = /usr/bin/scp

# ssh key file
key = /home/od/ec2-keys/id_odkey

# remote user
user = root

# cluster definition
# cluster.<name> = <node0> <node1> ...

cluster.ec2 = \
  ec2-174-129-169-130.compute-1.amazonaws.com \
  ec2-75-101-218-88.compute-1.amazonaws.com    \
  ec2-75-101-218-146.compute-1.amazonaws.com \

Olaf David
nc2 Examples

$nc2 -v -c ec2:0-4 -exec ls -l

$nc2 -v -c ec2:0 -exec tc-start-server.sh

$nc2 -c ec2:1-* -put /model.jar /tmp/model.jar

$nc2 -c ec2:1-5 -exec java -jar model.jar
TW EC2 EXAMPLE
Thornthwaite Water Balance Model (Thornthwaite, 1948; Mather, 1978; 1979)
Modeling Workflow

1) Instantiate ngmf cluster on EC2

2) Fetching cluster information, configure nc2
3) Copy model files to all nodes
4) Start the TC server on node 0
5) Execute model on all nodes
6) Get results
Start NGMF AMI
Running instances
Fetching Cluster IPs

(aws_to_c3.sh)
#!/bin/bash
echo "cluster.default = \"
aws din | grep running | awk -F "|" '{ print $6 "\" }'
echo " "

$ ./aws_to_c3.sh
cluster.default = \
  ec2-174-129-169-130.compute-1.amazonaws.com \
  ec2-75-101-218-88.compute-1.amazonaws.com \
  ec2-75-101-218-146.compute-1.amazonaws.com \

$ ./aws_to_c3.sh >> ~/nc2.conf

Olaf David
TC Server Configuration

- NGMF TC configuration file (tc.xml)
  - Server host attribute customization
  - public DNS name
  - needed for JMX

```xml
<servers>
  <!-- Tell DSO where the Terracotta server can be found. -->
  <server host="ec2-174-129-169-130.compute-1.amazonaws.com">
    <data>%({user.home})/terracotta/server-data</data>
    <logs>%({user.home})/terracotta/server-logs</logs>
  </server>
</servers>
```
Copy model files

Put these on all nodes in cluster ‘ec2’

$nc2 -c ec2:0-* -put ngmf.jar ngmf.jar
$nc2 -c ec2:0-* -put ngmf.models.jar ngmf.models.jar
$nc2 -c ec2:0-* -put climate.cst climate.cst
$nc2 -c ec2:0-* -put tc.xml tc.xml

• NGMF jar
• Model jar
• Climate input file
• TC configuration
Staring the TC Server

Run on the first node in cluster ‘ec2’

$nc2 -c ec2:0 -exec /opt/tc/bin/start-tc-server.sh -f /root/tc.xml
TC Server Admin console

- Connect to running TC server(s)
  - Throughput analysis
  - Shared Data object content
  - ...

private EC2 IP

public EC2 IP
Running the model on all nodes

Time the overall cluster execution

Run on nodes 1..max in cluster ‘ec2’

$ time nc2 -c ec2:1-* \ 
  -exec /opt/tc/bin/dso-java.sh \ 
  -cp 'ngmf.jar:ngmf.models.jar' \ 
  -Dtc.config=/root/tc.xml \ 
  tw.ThornthwaiteMC
Instrumentation for NAM & synchronization (tc.xml)

...
<dso>
  <roots>
    <root>
      <field-name>tw.ThornthwaiteMC.app</field-name>
    </root>
  </roots>

  <instrumented-classes>
    <include>
      <class-expression>tw.ThornthwaiteMC</class-expression>
    </include>
    <include>
      <class-expression>models.CompoundExecutor*</class-expression>
    </include>
  </instrumented-classes>

  <locks>
    <autolock>
      <lock-level>write</lock-level>
      <method-expression>* *.*.*(..)</method-expression>
    </autolock>
  </locks>
</dso>
...
Model Output

[ec2-75-101-225-141.compute-1.amazonaws.com] Starting BootJarTool...
[ec2-174-129-167-37.compute-1.amazonaws.com] Starting BootJarTool...
[ec2-174-129-175-95.compute-1.amazonaws.com] Starting BootJarTool...
[ec2-174-129-175-95.compute-1.amazonaws.com]
[ec2-174-129-175-95.compute-1.amazonaws.com] Starting Terracotta client...
[ec2-75-101-225-141.compute-1.amazonaws.com]
[ec2-174-129-167-37.compute-1.amazonaws.com]
[ec2-174-129-167-37.compute-1.amazonaws.com] Starting Terracotta client...
[ec2-75-101-225-141.compute-1.amazonaws.com]
[ec2-174-129-167-37.compute-1.amazonaws.com] Starting Terracotta client...
[ec2-75-101-225-141.compute-1.amazonaws.com] Submit 19
[ec2-174-129-167-37.compute-1.amazonaws.com] Submit 18
[ec2-174-129-175-95.compute-1.amazonaws.com] Submit 17
[ec2-75-101-225-141.compute-1.amazonaws.com] Submit 16
[ec2-174-129-167-37.compute-1.amazonaws.com] Submit 15
[ec2-174-129-175-95.compute-1.amazonaws.com] Submit 14
[ec2-174-129-167-37.compute-1.amazonaws.com] ...
Stopping the server

Run on the first node in cluster ‘ec2’

$nc2 -c ec2:0 -exec ./stop-tc-server.sh -f /root/tc.xml
NATIVE CODE ACCESS
Java Native Architecture

• Open source project
• Simplify native code integration
  – Generic dynamic dispatch
  – No bridging code (JNI) required
  – Direct call from Java -> DLL (Shared Object)
  – Direct data type mapping
• 32 and 64 bit, all major OS
A F95 Subroutine

! File:   ftest.f90
! Author: od

FUNCTION potET(daylen, temp, days) BIND(C, name='hamon') ! 1)
   REAL*8,VALUE :: daylen,temp
   INTEGER*4,VALUE :: days
   REAL*8 :: potET
   REAL*8 :: Wt,D2

   Wt = 4.95 * exp(0.062 * temp) / 100.0
   D2 = (daylen / 12.0) * (daylen / 12.0)
   potET = 0.55 * days * D2 * Wt
   print *, Wt
   if (potET <= 0.0) then
      potET = 0.0
   endif
   if (temp <= -1.0) then
      potET = 0.0
   endif
   potET = potET * 25.4
END

1) F03 C name binding as ‘hamon’ (no underscores)
2) Argument passing as value
Example in NGMF

// Interface binding
interface ETLib extends Library {    // 1)
    double hamon(double daylen, double temp, int days);
}

// Binds the native library 'F90Lib.dll' to 'ETLib'     // 2)
ETLib etlib = (ETLib) Native.loadLibrary("F90Lib", ETLib.class);

@Execute
public void execute() {    // 3)
    int month = currentTime.get(Calendar.MONTH);
    potET = etlib.hamon(daylen, temp, DAYS[month]);
}

1) Define a Java interface for all DLL functions that should be accessible. Can be a subset of available DLL functions.
2) Bind the external DLL to the Java interface.
3) Call the dll using the binding object, pass in java data