The Cupid Integrated Development Environment for Earth System Models

Feature Overview and Tutorial

Rocky Dunlap[1]
College of Computing
Georgia Tech
rocky@cc.gatech.edu

April 10, 2014

[1]This work supported by the NASA CMAC program.

Contents

Chapter 1

Overview of Features

Cupid is a set of development tools to facilitate the adoption of geoscience modeling frameworks into new and existing model codebases. It features a framework-aware code editing environment and a cloud-based configuration tool to simplify configuring the compile and execution environment. The target framework is the Earth System Modeling Framework (ESMF) and its interoperability layer called the National Unified Operation Prediction Capability (NUOPC), which is currently being implemented in most major climate and weather models in the US. Cupid tools are intended for model developers who have prior experience with model development workflows, but are new to developing with ESMF and NUOPC. It is also aimed at developers interested in exploring the benefits of using the Eclipse Integrated Development Environment (IDE) for improving development productivity.

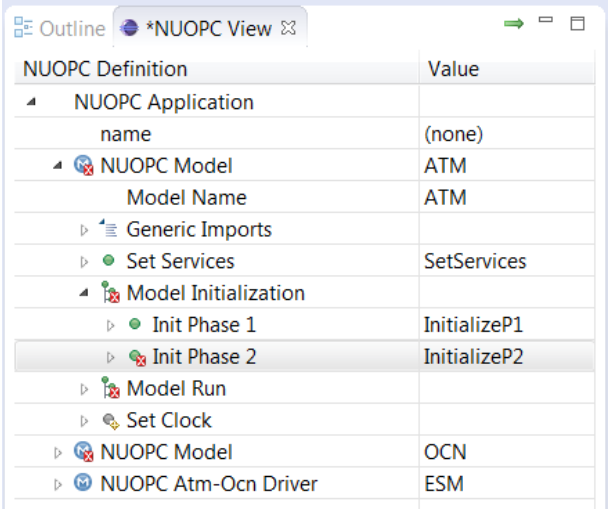
Use of modeling frameworks is quickly becoming the norm for both operational and research climate and weather models. Modeling frameworks provide a number of benefits including mechanisms for componentizing complex codebases, functions and data structures for coupling independent models into a single simulation, increased developer productivity through code reuse, improved quality and robustness of features compared with "home grown" solutions, and fast execution via parallel data transfer and interpolation operators.

In a framework-based application, such as a coupled model that uses ESMF, some application behaviors are provided by the framework and some are provided by the application developer. For example, ESMF provides functions for transferring and interpolating field data from one model's native grid to another model's native grid. However, ESMF does not prescribe entirely what it means for the model to take a step forward in time since that behavior is application specific. A framework provides a set of abstractions, *framework-provided concepts*, that the developer is required to instantiate and configure in their code. Creating a framework-based application is called *framework completion* because the developer fills in application behaviors not provided by the framework, or specializes behaviors provided by the

framework. Software engineering research has shown that even for well-designed frameworks, writing correct framework completion code is difficult because it requires a deep understanding of the framework's behavior.

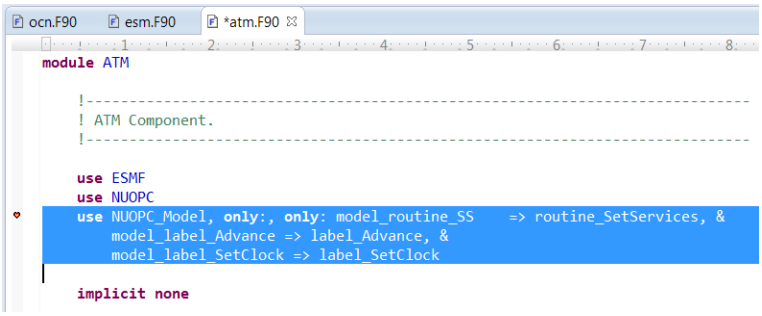
The Cupid tools adds framework-specific intelligence to the Eclipse Integrated Development Environment in order to facilitate adoption of ESMF and NUOPC. The features include:

- A tool for **reverse engineering** an existing codebase to determine what ESMF and NUOPC framework concepts are present in the code. The reverse engineering function does not require execution of the user's code—instead, it operates during the phase of development when code is written, such as when framework code is first introduced into an existing model. The reverse engineered model is presented to the user alongside the source code in the form of a tree where nodes correspond to framework concepts. Clicking on a node brings up the relevant code fragments in the code editor. The reverse engineering tool also checks for code-level compliance to NUOPC technical rules and offers suggestions for addressing compliance issues.

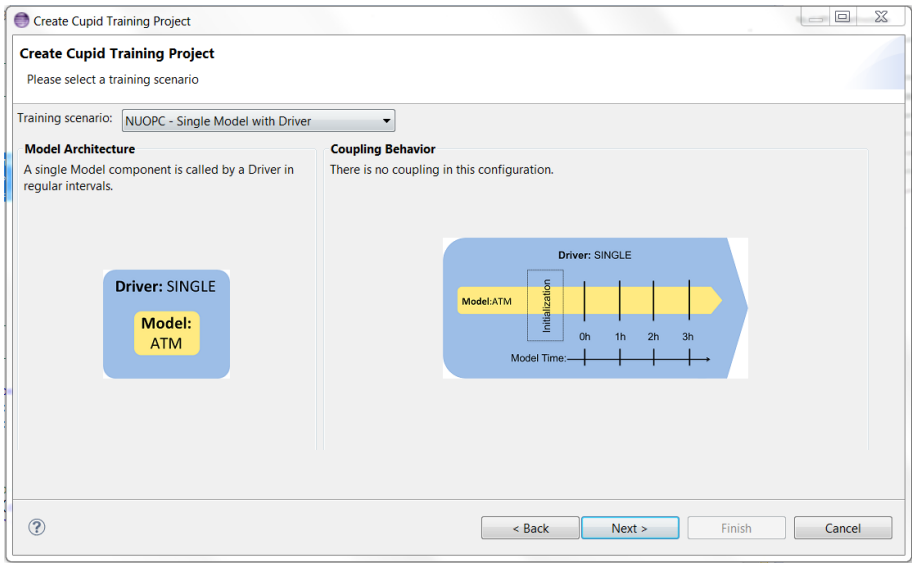


- A tool for **automatic source code generation** of NUOPC-compliant code fragments. The generated code can often be used as is, although the tool does not prevent further customization of the generated code when required. The generated code is woven into the user's existing code at the appropriate places, keeping the existing code structure intact. The code generation feature helps the developer understand what framework code is required and where it should be located. For example, the tool generates variable declarations for framework-specific types, calls to framework functions, and skeletons for

callback subroutines that the developer must implement. The code generation feature also works hand-in-hand with the reverse engineering function: after code fragments have been generated and (optionally) customized, the codebase can be reversed engineered again to update the outline and check for any compliance issues.



- A **cloud configuration** feature that allows the user to select a training scenario and, within a few minutes, configure, compile, and execute both skeleton models and realistic models on virtual machine instances using the Amazon EC2 platform.



Chapter 2

ESMF and NUOPC

This section describes the Earth System Modeling Framework (ESMF) and the National Unified Operational Prediction Capability (NUOCP) and provides references for those interested in finding out more. Readers already familiar with ESMF and NUOPC may choose to skip this section.

2.1 The Earth System Modeling Framework

ESMF is a high-performance software framework designed for numerical geoscience models. Some of the framework-provided concepts include model components (`ESMF_GridComp`) and coupler components (`ESMF_CplComp`; mediators between model components), and data types for model state (`ESMF_State`), distributed arrays (`ESMF_Array`), physical fields (`ESMF_Field`), and numerical grids (`ESMF_Grid`; discretization schemes). An ESMF-based application is typically designed as a hierarchy of model components where components communicate by exchanging `ESMF_State` objects via framework-provided interfaces. `ESMF_GridComp`s and `ESMF_CplComp`s have user-customizable `initialize()`, `run()`, and `finalize()` methods. For more information about ESMF, see the ESMF User's Guide and the ESMF Reference Manual.

2.2 The National Unified Operational Prediction Capability

To promote interoperability of model components, NUOPC is a set of generic components, metadata conventions, and behavioral protocols encoded in a software layer on top of ESMF. Together, these elements form the basis of a *common model architecture*—a standard way of building models in order to make it easier to assemble coupled models using components from different sources. NUOPC is currently being implemented in research and operational models such as the HYCOM ocean

model [**?**], GFDL's MOM5 ocean model [**?**], and NASA's ModelE climate model [**?**]. Additional information about NUOPC can be found on the NUOPC home page.

NUOPC applications are built by combining four basic building blocks called *generic components*. The four types of generic component are `Driver`, `Model`, `Mediator`, and `Connector`. Many component behaviors have been predefined by NUOPC. However, in some cases, the developer needs to provide implementations of behaviors not defined by NUOPC. Additionally, if the generic behavior does not meet the requirements of the coupled model, the developer may need to override existing behaviors. In both cases, the developer's implementation is typically provided in subroutines which are registered with and called by the framework. The process of providing new behaviors or overriding existing ones is called *specialization*. As defined here, *specialization* is conceptually similar to how a class overrides a parent class method to provide a different implementation in an object-oriented programming language. However, because the public ESMF and NUOPC APIs are not implemented in an object-oriented language, a custom specialization mechanism has been defined. Understanding the specialization process is essential for adopting NUOPC into a model's codebase.

The `Driver` generic component implements a harness of ESMF components and `ESMF_State` objects and it is specialized by plugging in `Model`, `Mediator`, `Connector`, and other `Driver` components. The `Driver` initializes its child components according to an *Initialize Phase Definition* and drives their `run()` methods according to a *Run Sequence*. `Model` wraps a user's code so it can be plugged into a `Driver`. `Model`s represent major geophysical domains such as atmosphere, ocean, and ice. `Connector`s and `Mediator`s manage communication between `Model`s. `Connector`s implement standard interactions such as parallel redistribution or regridding (interpolation) of fields and `Mediator`s implement complex `Model` interactions requiring customized code. Figure 2.1 illustrates several possible architectural configurations of NUOPC components.

To take full advantage of NUOPC, developers must ensure that model components comply with NUOPC architectural constraints and technical rules. The full definition of NUOPC compliance is available on the NUOPC compliance web page.

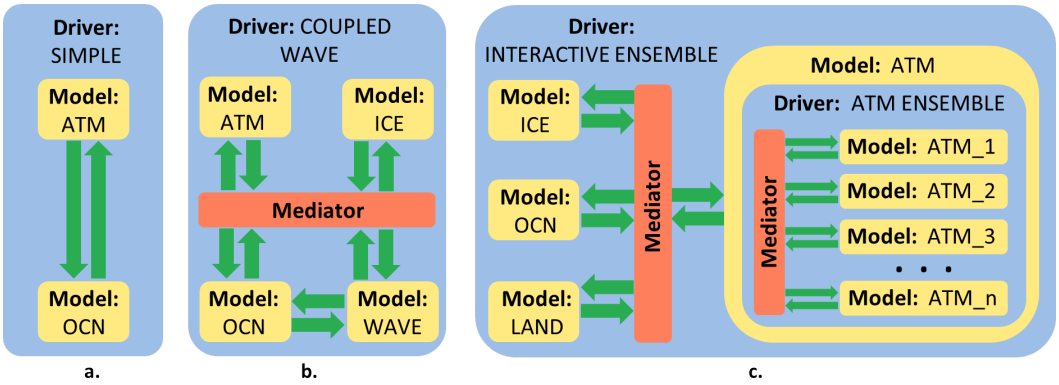


Figure 2.1: **a.** A `Driver` (blue box) with two child `Models` (yellow boxes) and simple `Connectors` (green arrows). **b.** A configuration in which a `Mediator` (orange box) couples atmosphere, ocean, ice, and wave `Models`. **c.** A complex configuration showing nested `Drivers`.

Chapter 3

Installation

3.1 Download and Install the Cupid Plugin

This section describes how to install the Cupid tools. Cupid is a plugin for the Eclipse Integrated Development Environment (IDE) and is available on all its supported platforms (Windows, Mac, Unix).

1. **Download and install Eclipse for Parallel Application Developers, version 4.3.2 SR2 (Kepler)**.

   The main download page is: http://www.eclipse.org/downloads/.

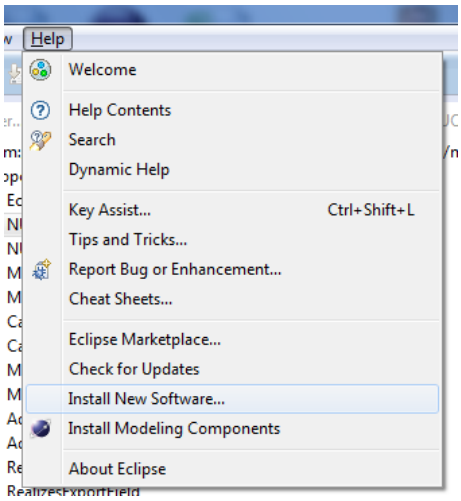
   There is a list of available Eclipse packages. Be sure to choose "Eclipse for Parallel Application Developers" as it will come pre-bundled with the necessary plugins for working with remote systems.

2. **Unpack the downloaded file into a local directory and run Eclipse by double clicking on the Eclipse executable.**

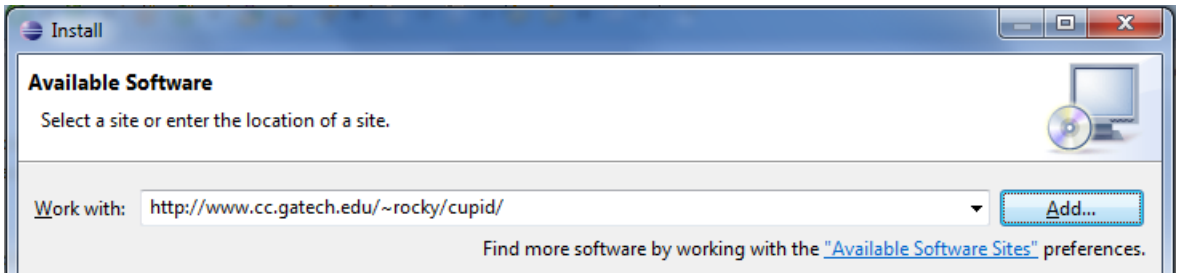
   The first time you start Eclipse, you will be prompted to select a location for your workspace. Choose an empty folder.

3. **Install the Cupid Plugin from the Cupid Update Site.**

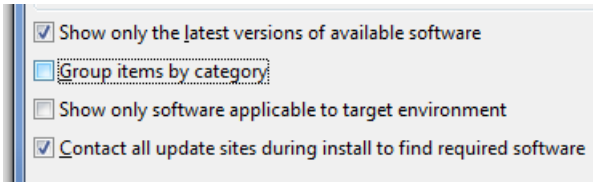
   a) Click Help → Install New Software



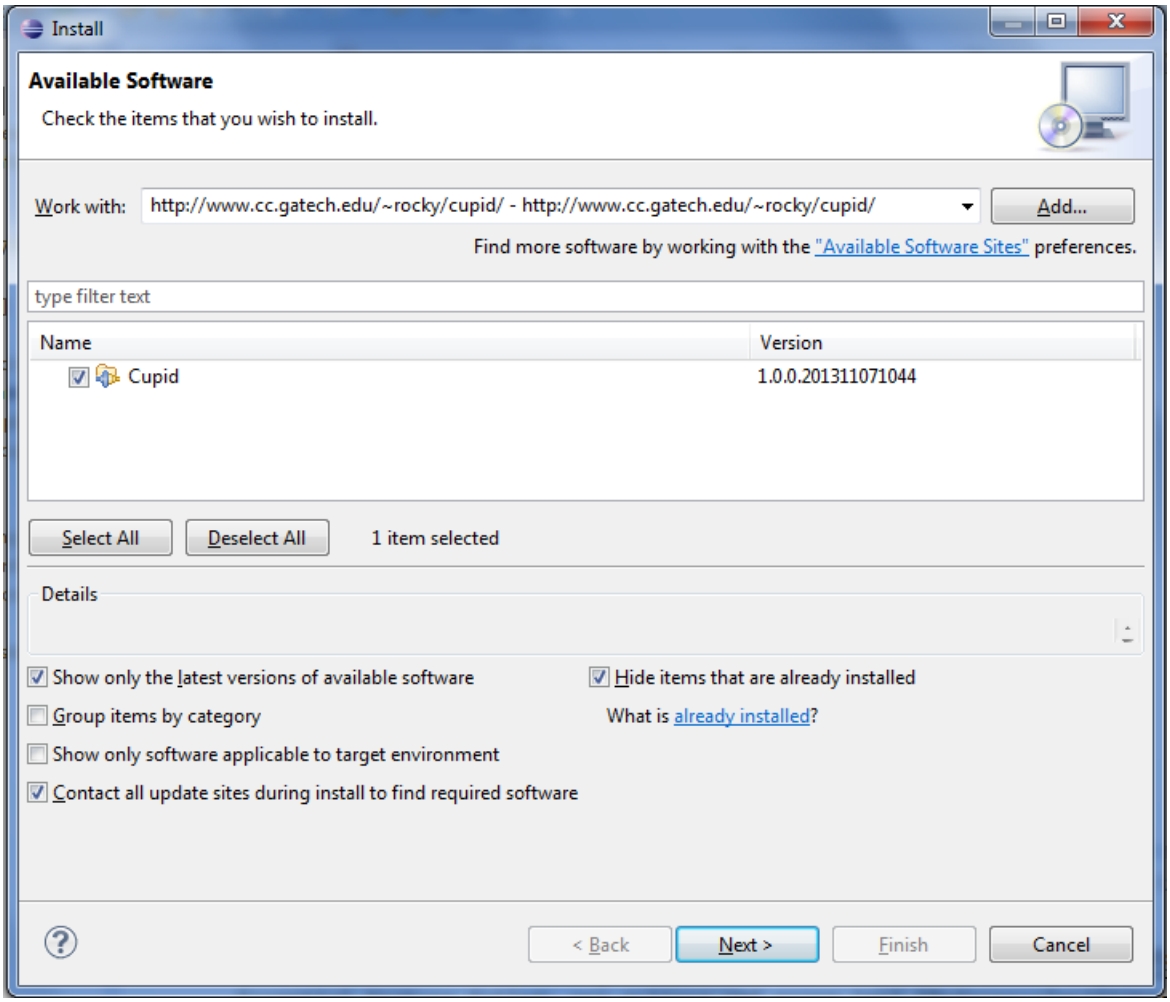
b) Put the Cupid Update Site URL into "Work with..." You will be prompted to give the update site a name of your choosing.
The Update site URL is: `http://www.cc.gatech.edu/~rocky/cupid/`



c) Uncheck the "Group items by category" option.

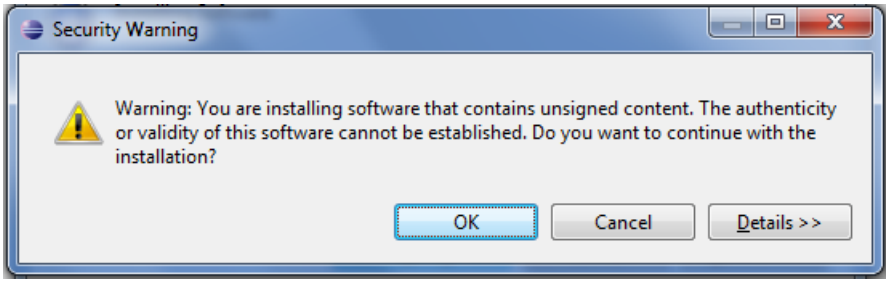


d) Select "Cupid" from the list and click Next.



e) You will need to click Next a couple more times and accept the license agreements. Then click Finish. The Cupid plugin and its dependencies will be downloaded and installed.

During the process, you may receive a message that the software contains unsigned content. Click OK.



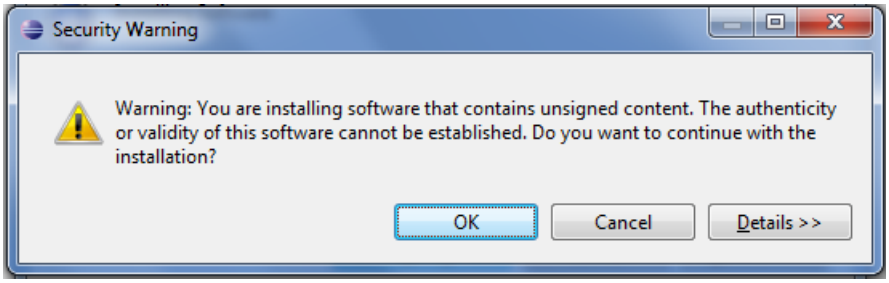
f) After installation, you will be prompted to restart Eclipse. Click Yes.

3.2 Configure Cloud Account Credentials (Optional)

Cupid can create cloud-based virtual machine instances to serve as the computational environment for training scenarios. If you plan on using the cloud configuration feature of Cupid, you need to set up your cloud credentials the first time you run Cupid. Currently, the supported cloud provider is Amazon EC2. You may use your own credentials or request access to the NESII cloud.

(Note: Cupid automatically launches Amazon machine instances when you create a new training project. Alarms are set up to automatically kill instances after 50 minutes of idle CPU utilization. However, it is your responsibility to ensure that any unused instances are terminated to avoid unnecessary cloud computing charges to your account.)

1. **In the Eclipse menu, select Window → Preferences. Then select Cupid Preferences in the list on the left.**
2. **Enter your Amazon Web Services (AWS) access key and secret key.**
3. **Click OK.** Your credentials have now been set up.



Warning: Only set up AWS credentials on a secure, private machine as they are currently stored in plain text in the Eclipse configuration metadata. This feature of Cupid is currently in prototype mode with minimal consideration of security ramifications. It is highly recommended to set up an AWS user account with limited privileges for use with Cupid. For more information, see the AWS Identity and Access Management documentation.

Chapter 4

Cloud-based Training Environment

Cupid simplifies the process of configuring a computational environment capable of compiling and executing high-performance geoscience models. IDEs package a lot of development tools into a single application to help manage and simplify the software development workflow. Although IDEs aim to increase developer productivity, they can still introduce a steep learning curve. Some challenges with using IDEs for geoscience model development include:

- Understanding the basic steps involved in moving from source code to a running model
- Making sense of the many development tools and features available in the IDE
- Setting up a high-performance computational environment capable of configuring, compiling and executing model code
- Configuring the IDE to connect to remote computational environments

Cupid's cloud integration feature allows a developer to select a training scenario and, within a few minutes, configure, compile, execute, and view the standard output of both skeleton models and realistic models. (This does not include data file post processing or plot generation.) This feature relies on a set of pre-configured machine images that can be instantiated on Amazon EC2 cloud infrastructure. The machine images contain all of the necessary software dependencies for the selected scenario. Furthermore, Cupid automatically configures the IDE to connect to and synchronize source code with cloud-based virtual machines.

While understanding how to set up and use a high-performance computational environment, including acquiring and configuring prerequisite software packages, is an important part of geoscience modeling, it should not preclude non-experts from

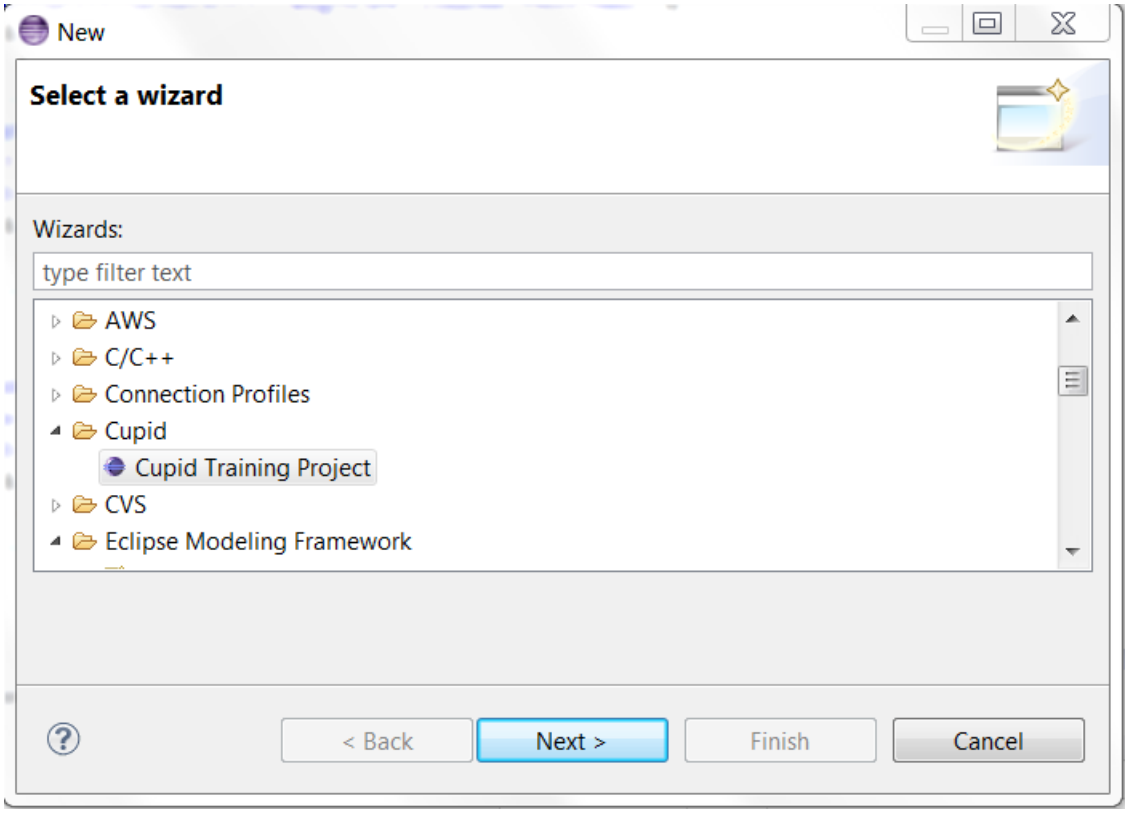
quickly setting up a complete computational environment and begin getting their feet wet with writing framework-based code.

4.1 Creating a new Cupid Training Project

This section of the tutorial describes how to use the *New Cupid Training Project Wizard* to create a new Eclipse project, populate it with a skeleton NUOPC application, and start up a remote virtual machine instance for compiling and executing the project.

Before beginning this section, please ensure that your Amazon EC2 credentials have been set up in the Cupid preferences. For instruction on how to do this, see section 3.2.

1. **Click File → New → Other and choose Cupid Training Project in the Cupid folder. Click Next.**

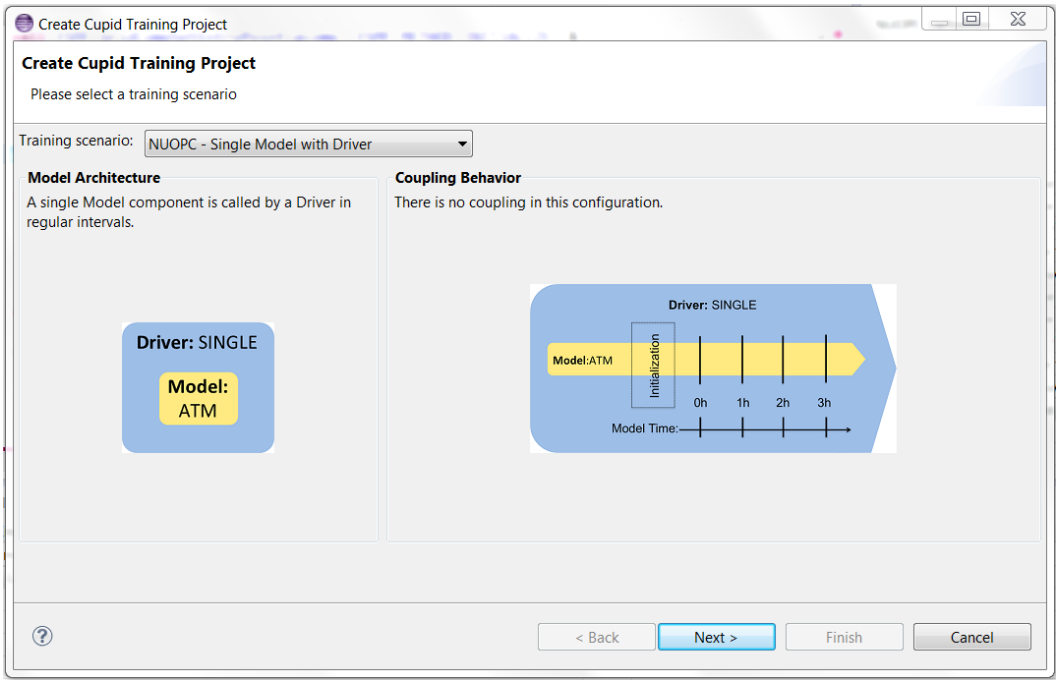


2. **On the first page of the wizard, choose a training scenario.**

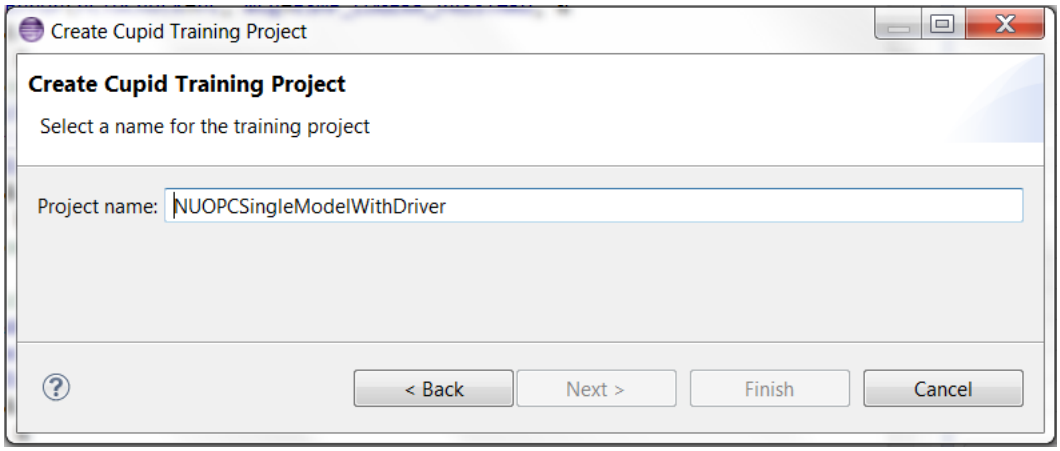
   - NUOPC — Single Model with Driver

- NUOPC — Coupled Atmosphere-Ocean Driver *(coming soon)*
- NUOPC — Coupled Atmosphere-Ocean with Mediator and Driver *(coming soon)*
- ModelE — Basic Configuration (EM20 rundeck) *(coming soon)*

Changing the training scenario in the list will update the screen to show the model architecture and coupling behavior of the selected scenario. **Choose the scenario "NUOPC Single Model with Driver" and click Next.**



3. **Choose a name for your project or accept the default name.** Note that project names must be unique, so you must choose a name that does not already exist in your workspace. **Click Next.**

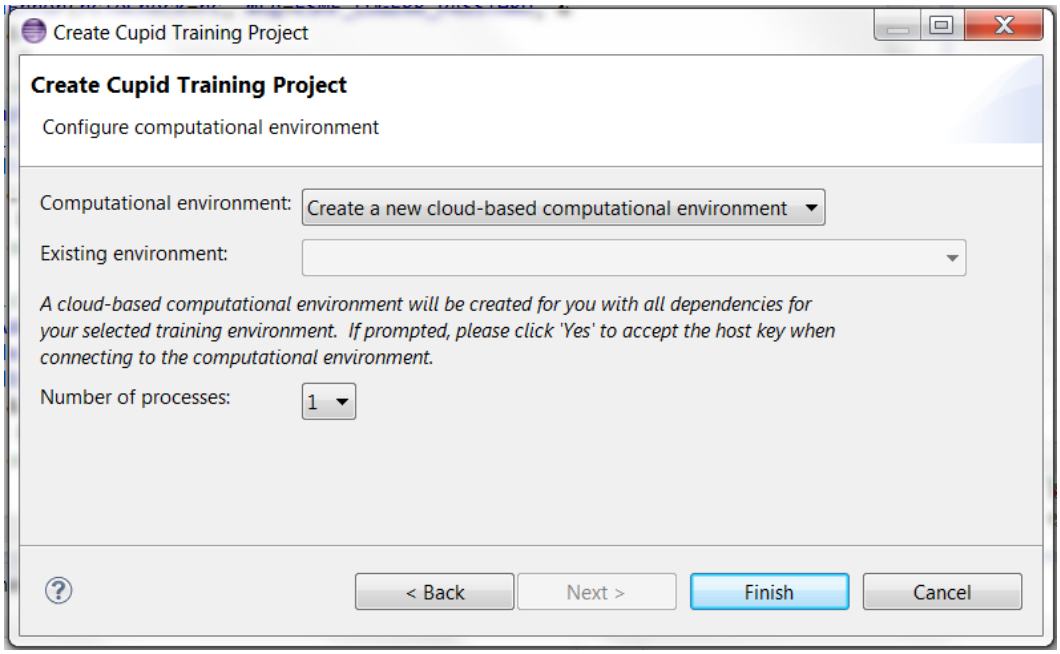


4. Choose a computational environment on which to compile and run the training scenario. The three options are:

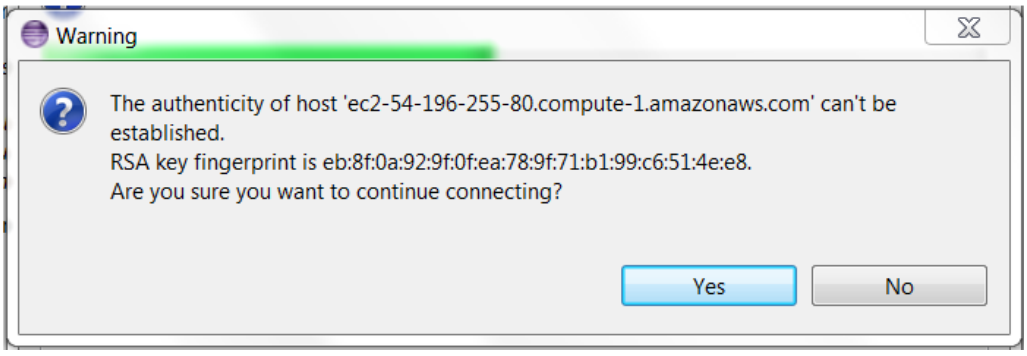
   - Create a cloud-based computational environment
   - Use my local machine
   - Use an existing remote environment *(this option is not always available)*

   The first option will create a preconfigured virtual machine instance for you with all dependent software (model source code, Fortran compiler, MPI, NetCDF, ESMF, etc.). This is the recommended option unless you know you have a supported environment already set up locally. In some cases, a third option will be available to re-use an existing remote environment that is already running.

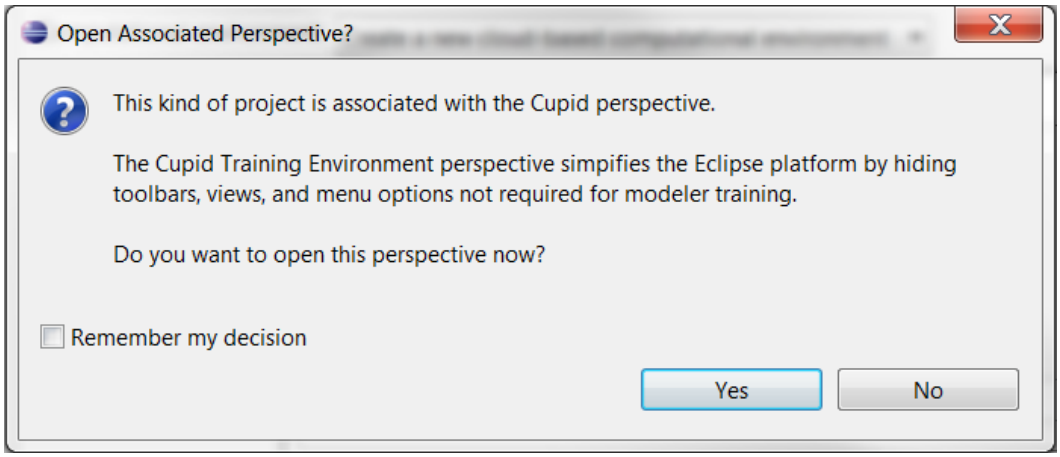
   Choose the first option and leave the number of processes at 1. Click Finish.



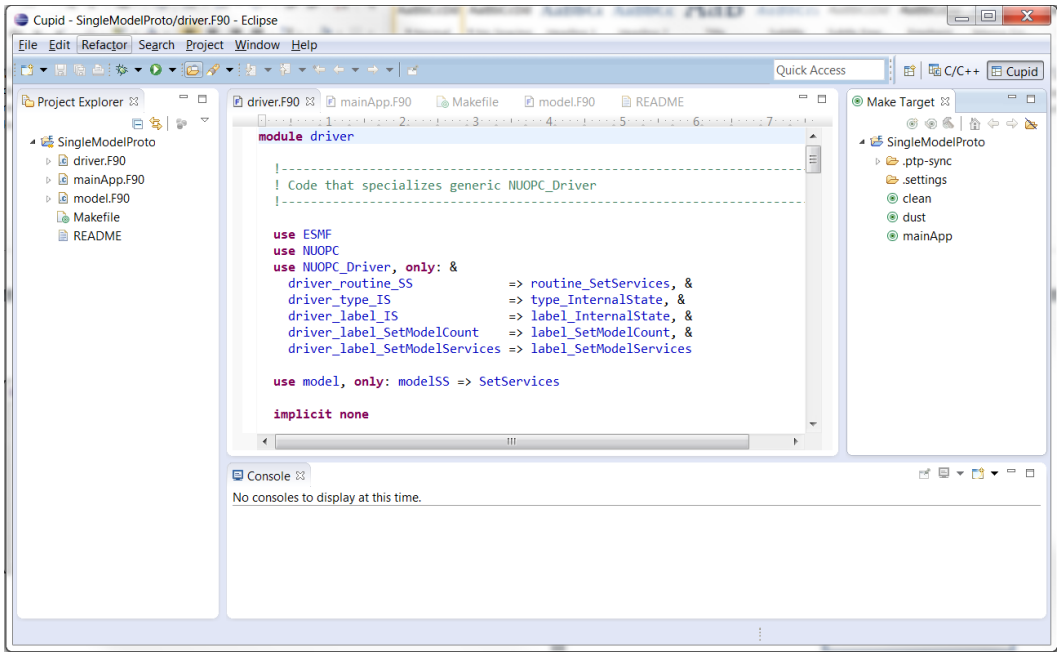
5. It will take up to several minutes for the new computational environment to start up. **During this process you will be asked to accept the host SSH key. Click Yes.**



You might also be asked if you would like to switch to the Cupid Perspective. An Eclipse perspective is a particular screen layout customized for specific tasks. The Cupid Perspective hides a number of Eclipse tools and commands that are not required for the training. **Click Yes to switch to the Cupid Perspective.**



6. You should now see your project in the Project Explorer on the left. The Cupid Perspective also exposes several other views including a Fortran source code editor, a Make Target view on the right for compiling your code and a Console at the bottom for viewing output from the compiler.

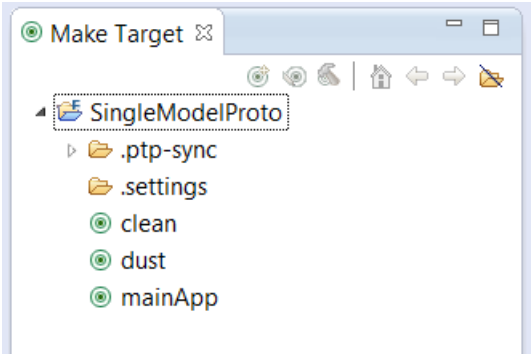


4.2 Compile and Run the NUOPC Single Model with Driver training scenario

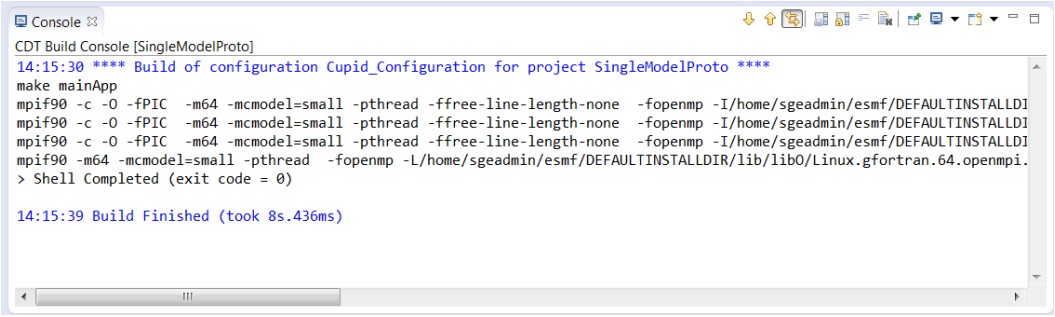
In this section of the tutorial, you will learn how to compile and execute the source code provided in the NUOPC Single Model with Driver training scenario. An

Eclipse Cheat Sheet is available for this task. (Click Help → Cheat Sheets and choose "Compile and Run a NUOPC Application" in the Cupid folder.)

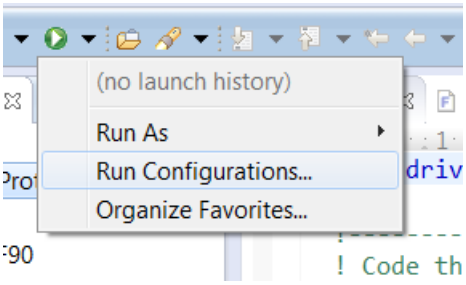
1. Ensure that the Make Target view is showing (see below). If not, **choose Window → Show View → Other... and select "Make Target" under the Make folder.** Open the folder in the Make Target view with the same name as the project you created.



2. If you used the New Cupid Training Project wizard to set up the training scenario, then the correct make targets have already been set up. To compile the NUOPC application, double click the "mainApp" target. You should be able to see the compiler output in the Console view at the bottom of the screen.



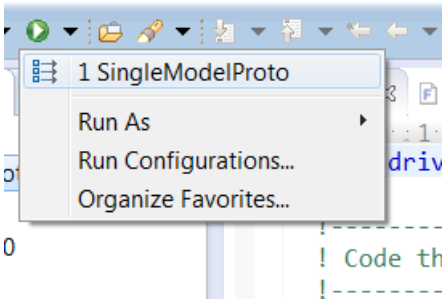
3. To run the compiled code, you must set up a run configuration. *(Note: This step will be automated in a future release of Cupid.)* Click the down arrow next to the Run As... button in the toolbar (green circle with a white arrow) and choose Run Configurations from the popup menu.



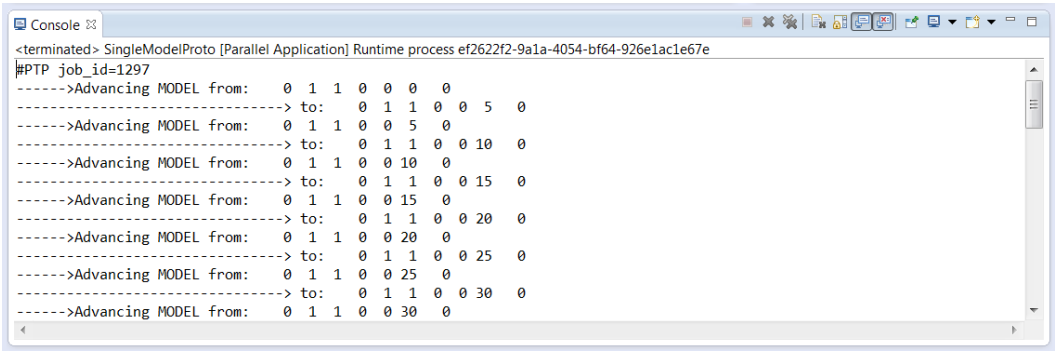
4. In the Run Configurations dialog, right click on Parallel Application and choose "New." Configure the run configuration as follows:

   - Resources tab
     - Target System Configuration: **Open MPI-Generic-Interactive**
     - Connection type: **Remote**
     - Connection: **Cupid Environment (Amazon EC2 ¡Project Name¿)**
   - Application tab
     - Project: should default to your project name
     - Application program: Click Browse and select "mainApp". The final path should be **/home/sgeadmin/SingleModelProto/mainApp**
   - **Click Apply then Run**

5. Once the Run Configuration has been created, you do not need to set it up again unless you need to change configuration settings. After running it the first time, the run configuration should be available in the Run Configurations dropdown list on the toolbar.



6. Standard output from the run will be shown in the Console view.


Console
<terminated> SingleModelProto [Parallel Application] Runtime process ef2622f2-9a1a-4054-bf64-926e1ac1e67e
#PTP job_id=1297
------>Advancing MODEL from: 0 1 1 0 0 0 0
--------------------------------> to: 0 1 1 0 0 5 0
------>Advancing MODEL from: 0 1 1 0 0 5 0
--------------------------------> to: 0 1 1 0 0 10 0
------>Advancing MODEL from: 0 1 1 0 0 10 0
--------------------------------> to: 0 1 1 0 0 15 0
------>Advancing MODEL from: 0 1 1 0 0 15 0
--------------------------------> to: 0 1 1 0 0 20 0
------>Advancing MODEL from: 0 1 1 0 0 20 0
--------------------------------> to: 0 1 1 0 0 25 0
------>Advancing MODEL from: 0 1 1 0 0 25 0
--------------------------------> to: 0 1 1 0 0 30 0
------>Advancing MODEL from: 0 1 1 0 0 30 0

Chapter 5

Reverse Engineering and Compliance Verification of NUOPC Applications

Most climate and weather model codebases are staggeringly large and obtaining an overview of the model, its subcomponents, and their interconnections is a cumbersome, time-consuming task. Often, this can only be accomplished by manually reading through the top-level source files to establish a mental picture of the overall structure of the model and its data flows.

Instead of viewing a model as an opaque, complex set of source files, Cupid's reverse engineering feature parses a model's source code and produces an abstract representation of the framework concepts that are present in the code. The reverse engineering feature is a kind of static analysis because the source code is analyzed before it is compiled and does not require execution of the model. This representation is shown to the user alongside the source code in an outline form called the NUOPC tree viewer. This provides an alternative, more abstract perspective for viewing a model's source code. Some of the main concepts provided by NUOPC are Drivers, Models, Mediators, and Connectors. If a codebase contains any of these generic components, the reverse engineering function will automatically find them and present them in outline form. This provides an architectural overview of an entire coupled system without requiring the developer to read through thousands of lines of source code.

NUOPC ensures interoperability of modeling components by specifying a set of technical rules that model implementers should follow. In addition to presenting a high level view of framework concepts in source code, Cupid's compliance checking feature provides feedback to the user when potential compliance issues are discovered in a reverse engineered model. For example, when developing a NUOPC Model component, certain subroutines must be implemented and registered with

the framework. If a required subroutine or its registration is missing, Cupid can identify the problem and annotate the outline view with icons indicating that some required code is missing. Moreover, this feedback is provided immediately to the user during model development thereby reducing the number of runtime failures and improving efficiency of the development process.

5.1 Acquiring NUOPC Prototype Applications

In order to explore Cupid's reverse engineering and code generation capabilities, you need to acquire the source code for at least one of the provided NUOPC prototype applications. A description of the available prototype codes is available at https://earthsystemcog.org/projects/nuopc/proto_codes. The prototype source code itself is available in a Subversion repository on Sourceforge:

```
http://sourceforge.net/p/esmfcontrib/svn/HEAD/tree/NUOPC/tags/ESMF_6_3_0r/
```

Assuming you have already installed Eclipse and the Cupid plugin (see section 3.1), there are several ways to acquire the NUOPC prototype code in your local workspace:

1. Start up a cloud-based training environment as described in section 4.1.
2. Download a snapshot from Sourceforge and import it into a new Eclipse project.
3. Connect Eclipse to the Sourceforge Subversion repository and check out NUOPC prototype code.

In this section we will describe how to connect Eclipse to the NUOPC prototype Subversion repository and how to check out one of the prototype applications into a new project in your Eclipse workspace.

1. Install the Subversion plugin for Eclipse (Subversive)

   First, ensure that the Subversive plugin, which provides access to Subversion repositories, is installed on your copy of Eclipse. To determine if it is already installed go to Window → Show View → Other... and look for a folder called SVN. If it is there, Subversive is installed and you can skip the rest of this step.

   If you need to install Subversive follow these steps:

   a) Select Help → Install New Software from the Eclipse menu

   b) In the Work With box, choose `Kepler - http://download.eclipse.org/releases/kepler`

c) In the list, choose *Subversive SVN Team Provider* under the *Collaboration* group

d) Click Next a couple times, accept the license agreement, and click Finish.

e) You will be prompted to restart Eclipse. Choose Yes.

f) After Eclipse restarts, choose Window → Show View → Other..., open the SVN folder and choose SVN Repositories

g) The first time you select the SVN Repositories view, the *Subversive Connector Discovery* dialog will appear. Select one of the latest SVN connectors (e.g., SVN Kit 1.8.3) and click Finish. Click Next a couple times, accept the license agreement, agree to installing unsigned content, click Finish and restart Eclipse. Subversive is now installed and ready to use.

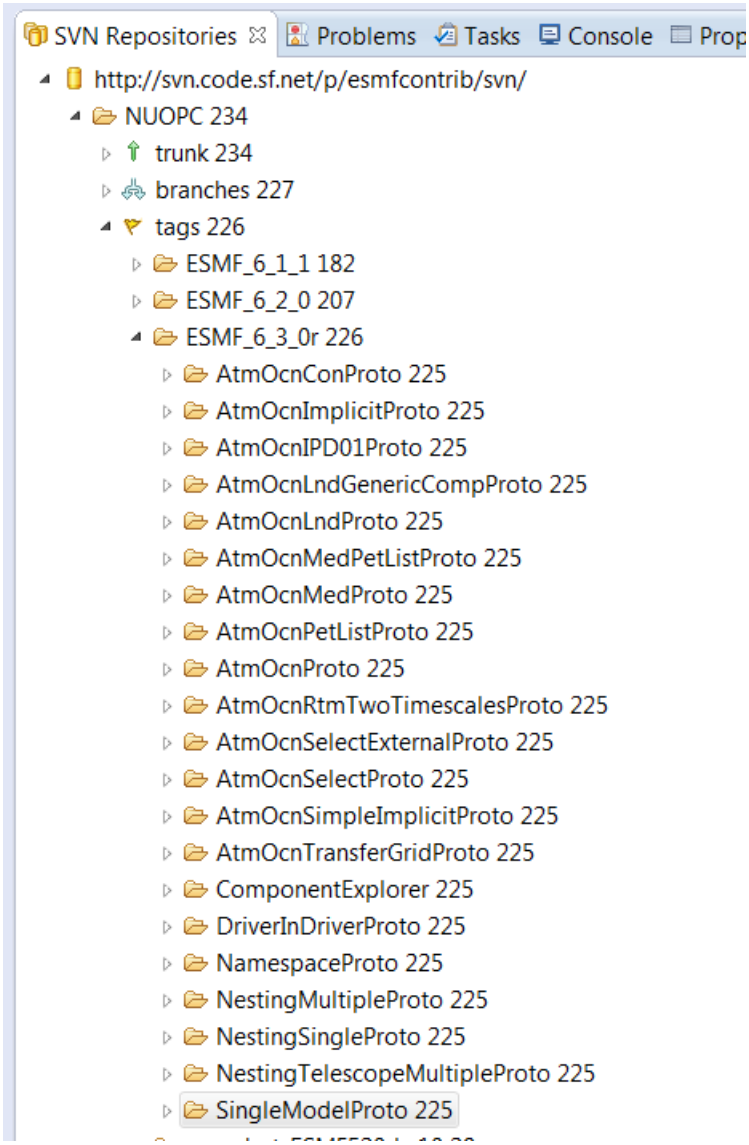
2. Add the NUOPC prototype codes repository to Subversive

a) In the menu, select Window → Show View → Other..., open the SVN folder and choose SVN Repositories.

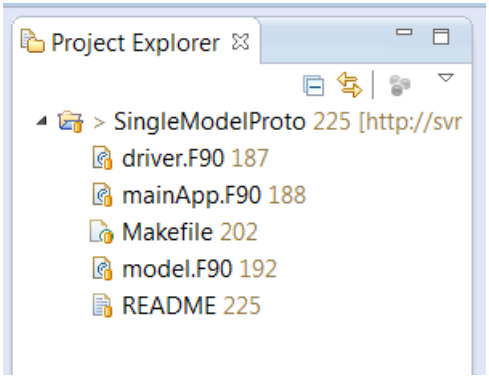
b) In the SVN Repositories view, right click and choose New → Repository Location...

c) In the URL field, put the URL of the NUOPC prototype codes Subversion repository: `http://svn.code.sf.net/p/esmfcontrib/svn/` and click Finish. The new repository location now appears in the list.

d) In the new repository location, navigate to NUOPC → tags → ESMF_6_3_0r. Each subfolder under that location is a separate NUOPC prototype application. Right click on the folder *SingleModelProto* and click Check Out.



e) You should now see a new project in your workspace named SingleModelProto.



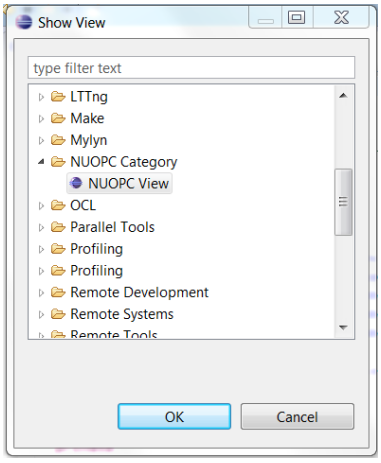
5.2 Reverse Engineer a NUOPC Prototype Application

In this section you will learn how apply Cupid's reverse engineering function to the *SingleModelProto* NUOPC application. You should now have a project called SingleModelProto in your Eclipse workspace containing several source files:

- driver.F90
- mainApp.F90
- Makefile
- model.F90
- README

1. Open the NUOPC tree view

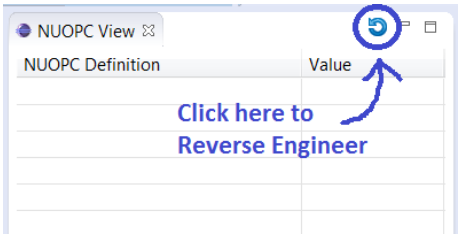
   From the Eclipse menu, choose Window → Show View → Other... and then select *NUOPC View* from the *NUOPC Category* folder.



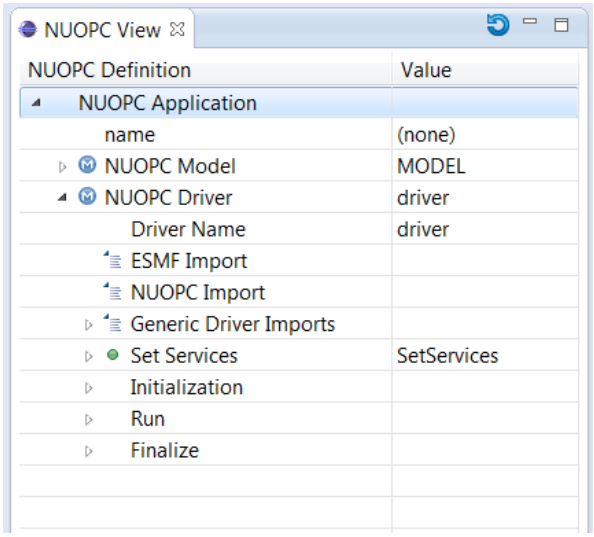
2. Open the file *driver.F90* from the SingleModelProto project by double-clicking the file in the Project Explorer.

3. With *driver.F90* open in the editor, click on the reverse engineer icon in the NUOPC tree viewer (circular blue arrow in the top right corner of the view).

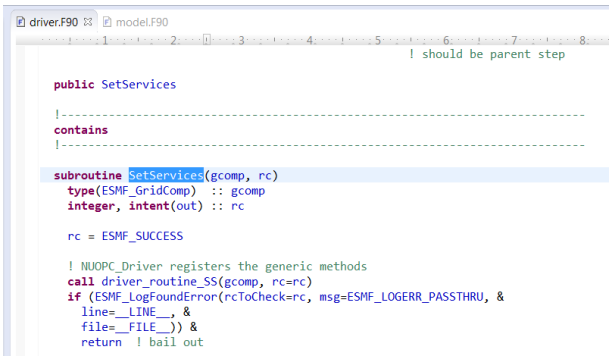
   The reverse engineering function works on one project at a time, not all projects in your workspace. The NUOPC tree viewer determines which project to reverse engineer based on the source file currently selected in the Project Explorer or open in the editor. Therefore, if you are working with multiple projects, be sure that a file from the project you want to reverse engineer is active in the editor.



4. The reverse engineered model outline now appears in the tree viewer. The elements in the reverse engineered model correspond to parts of the prototype application source code.

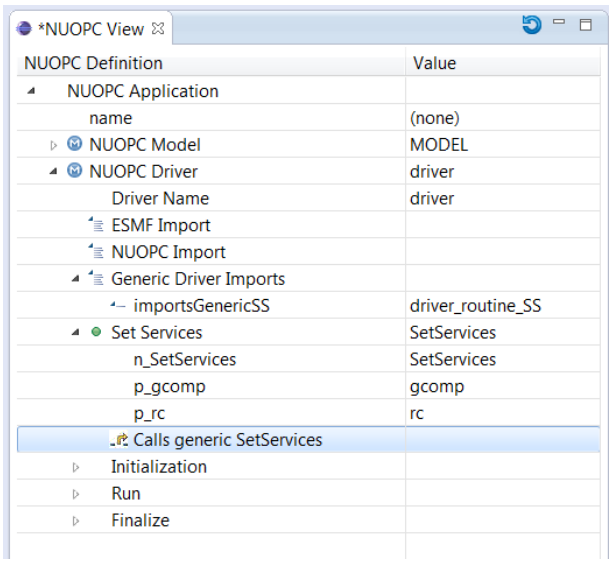


   Expand the *NUOPC Driver* element in the NUOPC viewer and find the *Set Services* element. The green circle indicates that this element maps to a Fortran subroutine. Double-click the *Set Services* element and the corresponding subroutine name will be highlighted in the source code editor.



Every NUOPC component is required to have a single public entry point called `SetServices`. This subroutine is called by a parent component or top-level program to register all of the execution phases and specialization points for the component.

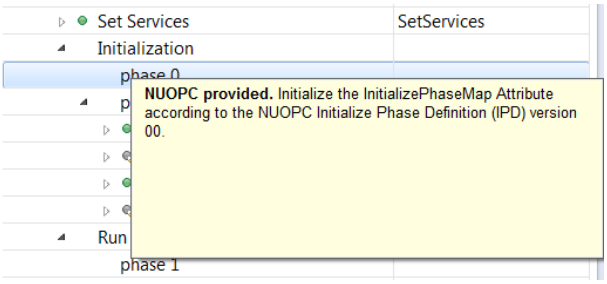
5. In the NUOPC tree view, find the element *Calls generic Set Services* under the *Set Services* element. The yellow arrow icon indicates that the element maps to a subroutine call found during reverse engineering. Double-click the element and the corresponding call will be highlighted in the source code.



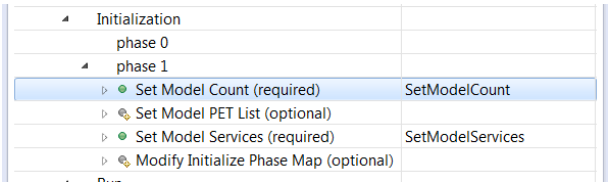
This call is an up call to the `SetServices` routine of the generic NUOPC Driver component and it handles registration of the methods common to all NUOPC Drivers. Calling the generic SetServices method is always required when implementing a specialized version of a NUOPC component (e.g., Model, Driver, or Mediator).

6. In the reverse engineered model, find the element *NUOPC Driver* → *Initialization* → *phase 0*. Note that it does not have an icon because this element

does not map to an element in the reverse engineered code. This is because the phase 0 initialization is handled by the generic NUOPC Driver and does not require a separate user-supplied implementation. The behavior of the generic implementation is described in a tooltip popup when the mouse hovers over the phase 0 element.



7. Hover over the *phase 1* initialization element in the reverse engineered model to see the documentation about this element. Again, this phase is provided by the generic NUOPC Driver although there are several ways in which the generic phase 1 behavior can be specialized. Two of the specializations are required, *Set Model Count* and *Set Model Services*, and two are optional, *Set Model PET List* and *Modify Initialize Phase Map*. Note that the two required specializations have been provided in the prototype code (indicated by the green subroutine icons) and the two optional specializations are not present (indicated by the grayed out icons).



5.3 NUOPC Compliance Verification

In this section, we will see how the Cupid reverse engineering tool can show compliance issues by identifying required source code elements that are missing. The compliance checking capabilities of Cupid are a companion to two other analysis tools, the *NUOPC Compliance Checker* and *Component Explorer*. The key difference lies in the kinds of analyses performed. At present, Cupid is purely a static analysis tool and identifies compliance issues by examining the abstract syntax of NUOPC application source code. The *NUOPC Compliance Checker* and *Component Explorer* support dynamic analysis, outputting compliance information in log form as the NUOPC application executes. As such these runtime tools can identify a greater number of compliance issues.

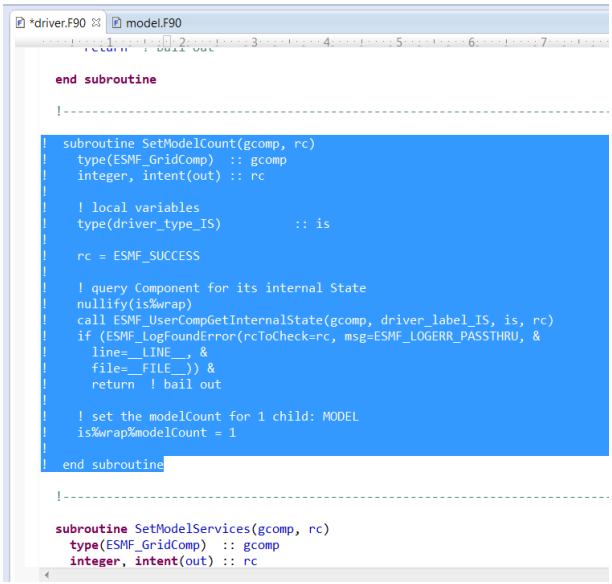
However, because Cupid's reverse engineering works on partially completed code, it is useful for identifying many compliance issues earlier in the development process, before any runs are attempted. The results of the static compliance check are available immediately as the NUOPC application is being developed and, as we will see in the next chapter, Cupid's code generation feature provides some additional assistance to the developer in bringing the code to a compliant state.

More information about the runtime compliance checking tools can be found in section 5 of the NUOPC Reference Manual.

In the steps that follow, we will create a compliance issue in the *SingleModelProto* application and see the results of the reverse engineered model.

1. Make sure you have the *SingleModelProto* source code in your Eclipse workspace.

2. Open the file *driver.F90* and find the subroutine `SetModelCount`. Select the entire subroutine, right click, and choose Toggle Comment. This will comment out the entire region of code.

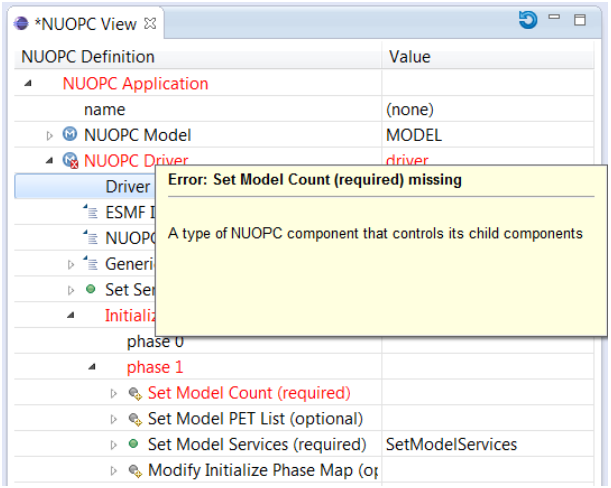


3. Save *driver.F90*.

4. In the NUOPC tree viewer, click the reverse engineering button again to re-run the reverse engineering function on the modified code. *(Currently, the tool does not automatically sync with updated code—the reverse engineering function must be run manually.)*

5. Elements in the reverse engineered model are colored red if a compliance issue is found at or below that element. Drilling down into the outline reveals the missing element: because the `SetModelCount` subroutine is commented

out, one of the required specialization points no longer appears in the code. Hovering over a red element shows a tooltip popup with a description of the compliance issue.



Chapter 6

Generating NUOPC-compliant Code

Even if a software framework is well designed, writing framework completion code is notoriously difficult, even for seasoned developers. Often, completing a single logical task requires making several code additions at multiple places spread throughout the application source code. If one or more of the required additions are inadvertently left out, the application may not behave as expected.

In the software engineering research community, many ideas have been proposed for how to help developers write framework-based applications correctly and efficiently. For ESMF and NUOPC, guidance is provided in the form of comprehensive API documentation (ESMF Reference Manual, NUOPC Reference Manual), system tests (included with source distribution), and small prototype codebases that show how to structure NUOPC applications based on the components in the modeled system (e.g., standalone atmosphere, coupled atmosphere-ocean, three-component system, etc.).

Cupid's code generation feature complements these static resources by generating on-the-fly NUOPC-compliant source code fragments directly in existing source files. The user initiates a code fragment generation by adding elements to a reverse engineered model in the NUOPC tree viewer. The source code is then synchronized with the tree viewer, generating the required code fragments. The generated code fragments can then be customized by the developer for their particular case. The following use case illustrates use of forward engineering feature:

> A developer has finished writing the initialization phases for a NUOPC Model component called ATM and now needs to add the capability to advance the model one time step. The developer right clicks on the ATM element in the NUOPC tree viewer and selects "Add Model Advance." Two things happen immediately: the tree viewer is updated with a new sub-element underneath ATM called "Model Advance" which in turn

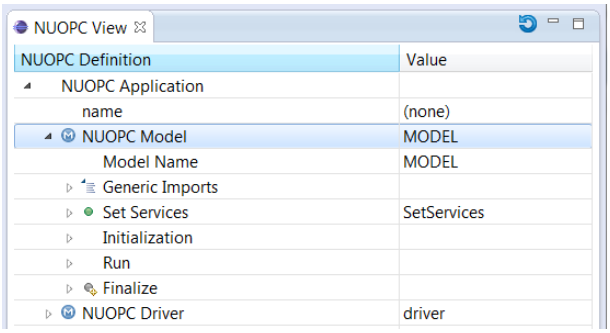
contains sub-elements "Registered in Set Services" and "Implementation." Then, source code fragments are generated inside the Fortran file for ATM including a call inside the ATM SetServices to register the Model Advance subroutine and a stub for the new Model Advance subroutine.

The use case shows some advantages of this approach compared to an approach in which code is copied-and-pasted from prototypical example code. First, the new Model Advance element added to the tree viewer included multiple sub-elements indicating that source code changes are required in at least two places: a new subroutine and a call to register this subroutine with the framework. This provides guidance to the developer to ensure that all framework requirements are met. The approach, therefore, is less error-prone than brute force copy and paste as some required code may inadvertently be left off. Also, the generated code fragments are customized based on the state of the existing source code. For example, the developer may be using specialized variable names. Since these variables have already been discovered during the reverse engineering phase, the generated code can reference these variables instead of requiring the developer to modify variables in copy-pasted code.

6.1 Generate a NUOPC Model Finalize method

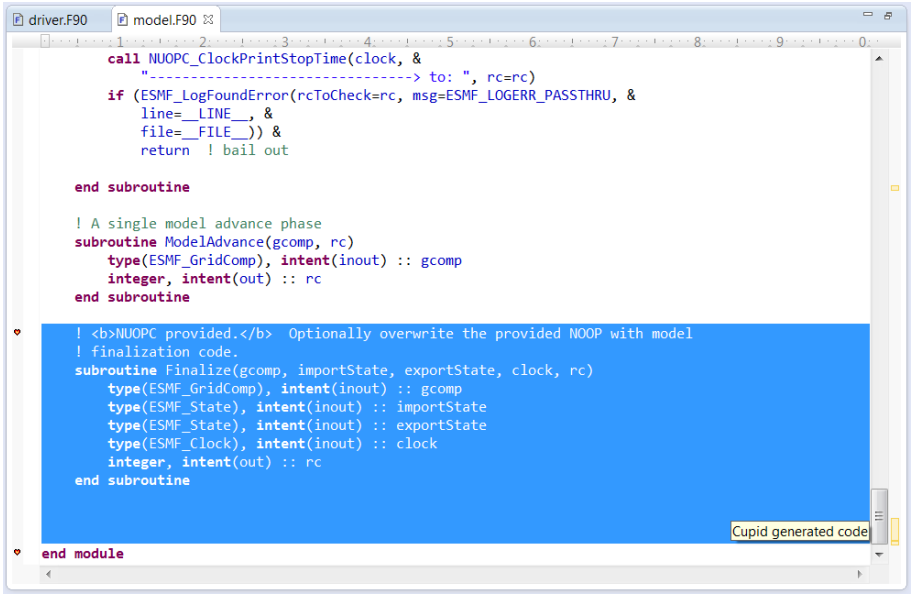
In this section we will demonstrate the code generation capabilities of Cupid by generating a subroutine stub for an optional specialization point and registering the subroutine in the `SetServices` method of the model.

1. Be sure you have the *SingleModelProto* project in your Eclipse workspace.
2. Open the file *model.F90*.
3. In the NUOPC view, click the reverse engineer button. The view should populate with the reverse engineered model.
4. Expand the *NUOPC Model* element and notice that the last child element, *Finalize*, has a grayed out icon. This indicates that there is no `Finalize` subroutine defined in *model.F90* (the subroutine is optional).



5. Right click on the *NUOPC Model* element in the outline and select *Add Finalize (basic)*. Both the NUOPC view and code editor will be updated.

   In the editor showing *model.F90*, two blocks of code are generated. The yellow blocks on the small vertical bar directly to the right of editor indicate which blocks of code were generated by Cupid. Clicking on a yellow block highlights the generated code, which has been inserted into the existing source file. The generated fragments of code are shown in listings 6.1 and 6.2



Listing 6.1: A call to register the generated subroutine as an ESMF Finalize method. This call is inserted at the end of the Model's `SetServices` method.

```
1 call ESMF_GridCompSetEntryPoint(gcomp, ESMF_METHOD_FINALIZE, &
2                                 userRoutine = Finalize, phase = 1, rc = rc)
```

Listing 6.2: A Finalize subroutine stub to be customized. The subroutine is inserted at the end of the module.

```
1  ! <b>NUOPC provided.</b> Optionally overwrite the provided NOOP with model
2  ! finalization code.
3  subroutine Finalize(gcomp, importState, exportState, clock, rc)
4                      type(ESMF_GridComp), intent(inout) :: gcomp
5                      type(ESMF_State), intent(inout) :: importState
6                      type(ESMF_State), intent(inout) :: exportState
7                      type(ESMF_Clock), intent(inout) :: clock
8                      integer, intent(out) :: rc
9  end subroutine
```

Chapter 7

The Behind-the-Scenes Meta-tool

This section describes theoretical and implementation aspects of the meta-tool used to define the mappings from framework-provided concepts to source code. This chapter may be of interest to software engineering researchers or those interested in using Cupid for defining their own framework-specific development tools. Reading this chapter is optional for users who wish to use the tool to develop NUOPC-based applications.

To support model developers in writing NUOPC-complaint code, the Cupid tool leverages existing work aimed at facilitating development of framework-based applications called Framework-Specific Modeling Languages (FSMLs). A FSML is a domain-specific language designed for a specific framework [1]. FSMLs are aimed at addressing some of the challenges involved in developing framework-based applications, especially knowing how to complete a framework correctly and how to ensure respect of its rules of engagement[1]. The language elements in the FSML's abstract syntax represent framework-provided concepts that the developer instantiates in code. FSMLs can be represented as a *feature model* [**?**], where features correspond to framework-provided concepts and each feature model configuration represents a valid framework completion.

Bidirectional mappings from framework-provided concepts to application source code are used to support forward and reverse engineering functions. In the forward direction, framework completion code is generated from an FSML instance—i.e., a Framework-Specific Model (FSM). In the reverse direction, an existing codebase is analyzed in order to recognize code patterns that correspond to framework concepts, producing a FSM. Taken together, the forward and reverse mappings enable round-trip engineering: the developer can seamlessly move between two perspectives, a zoomed in code-level perspective for customizing source code, and a higher level perspective showing the framework concepts present in the application and their

inter-relationships.

7.1 The NUOPC Framework-Specific Modeling Language

This section describes a subset of the abstract syntax of the NUOPC FSML. Figure 7.1 shows some of the framework-provided concepts in the NUOPC FSML represented as features in a feature model. The top-level concept node is `NUOPCApplication`. Its child features, `NUOPCModel`, `NUOPCDriver`, and `NUOPCMediator`, are the primary architectural components defined by NUOPC that require specialization by the developer. Subfeatures of `NUOPCModel` include `genericImports`, a container feature for required and optional module imports, `implementsSetServices`, which represents a framework-called subroutine that the developer implements, `initialization`, whose (elided) subfeatures represent the initialization subroutines that the developer implements, and `implementsModelAdvance`, which represents the developer-provided subroutine that advances the model forward in time.

Mapping definitions define the correspondence between features in the FSML and structural and behavioral code patterns. Cupid defines a set of mapping types for Fortran 90, the primary language binding supported by ESMF and NUOPC. Currently, only structural mappings types are supported, and they have been defined on an as-needed basis. Mappings are indicated in the figure next to each feature. `NUOPCModel`, `NUOPCDriver`, and `NUOPCMediator` each have a mapping definition of `module`. The top-level concept, `NUOPCApplication`, does not indicate a mapping. It will be implicitly mapped to an entire codebase. (Think of this as the root directory of a source tree.) Table 7.1 lists the current set of supported mapping types.

The mapping definition for the feature `implementsSetServices` is `subroutine: "#name(inout type(ESMF_GridComp) #gcomp, out integer #rc)"`. A subroutine signature includes the subroutine name, and optionally the intents (in/out/inout), types, and names of its formal parameters. Only subroutines with a matching name and matching argument intents and types are matched. In general, mapping definitions may include meta-variables that refer to features. The subroutine signature above contains three meta-variables, `#name`, `#gcomp`, and `#rc`. The `#name` meta-variable implicitly maps the subfeature `name` to the name of the mapped subroutine.

In the figure, some features are marked with a `!` indicating that they are *essential* features. A feature is only instantiated if all of its essential child features can be successfully mapped. For example, in Figure 7.1, the features `importsGenericSS` and `callsGenericSetServices` are essential. If their mappings fail, then the higher-level feature `NUOPCModel` will not be be instantiated.

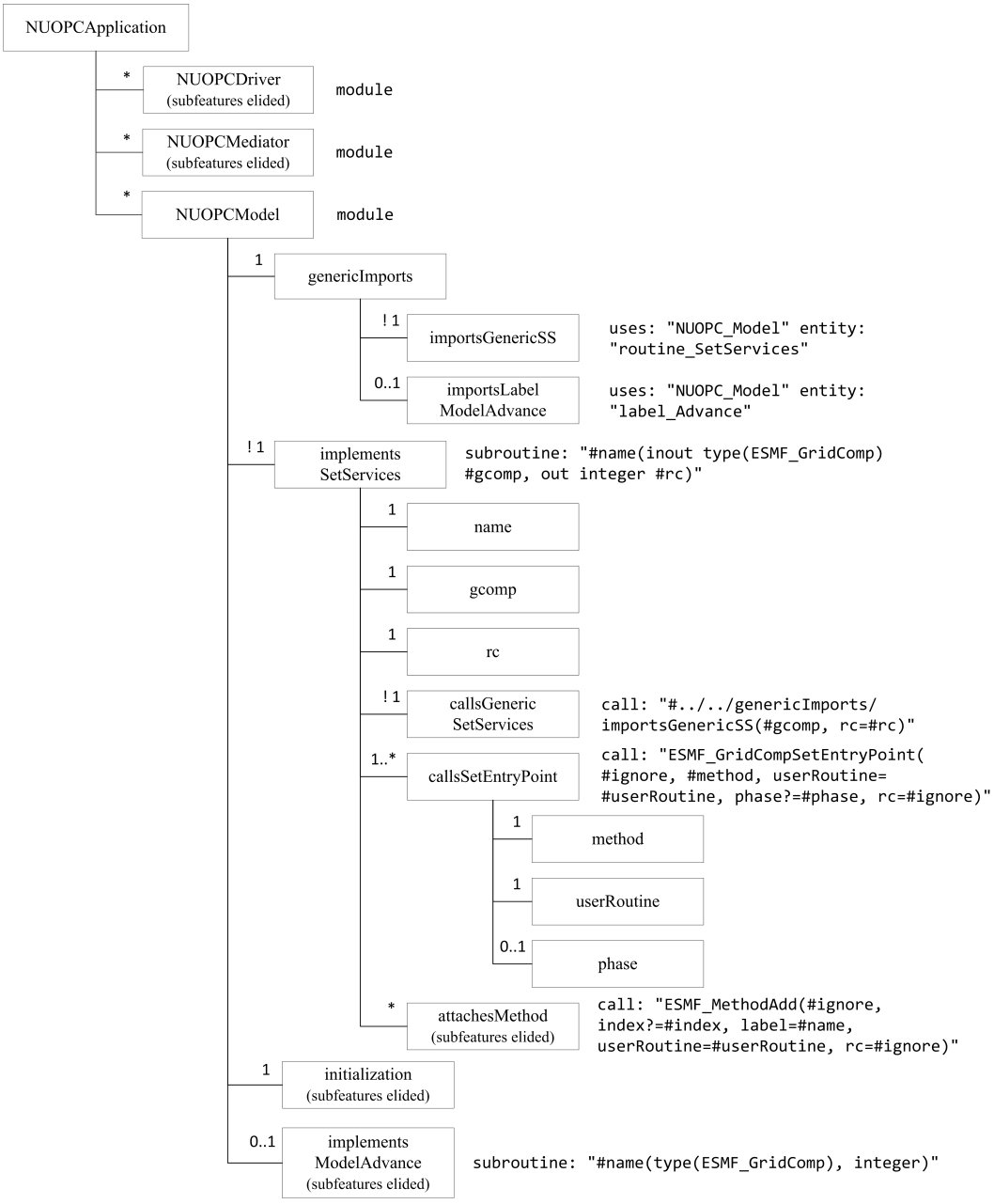


Figure 7.1: A partial feature model representation of the NUOPC FSML

Table 7.1: Mappings types for Fortran 90

Structural Pattern Expression	Structural Elements Matched
`module`	matches a Fortran module
`m moduleName`	matches the name of module `m`
`m usesModule: mn`	matches use statements in module `m` with imported module name `mn`
`u usesEntity: en`	matches import of entity named `en` within use statement `u`
`m uses: mn entity: en`	matches use statements in module `m` with imported module name `mn` and imported entity name `en`
`m subroutine`	matches subroutines defined within module `m`
`m subroutine: ss`	matches subroutines defined within module `m` with signature `ss`
`s subroutineName`	matches the name of subroutine `s`
`s formalParam: i`	matches the `i`th formal parameter of subroutine `s`
`s call`	matches calls with the implementation of subroutine `s`
`s call: cs`	matches calls with the implementation of subroutine `s` with call signature `cs`
`c argByIndex: i`	matches the `i`th actual parameter of call `c`
`c argByKeyword: k`	matches the actual parameter with keyword `k` of call `c`

7.2 FSML Implementation

In Cupid, a FSML is implemented as an Ecore model to take advantage of the Eclipse Modeling Framework (EMF) suite of tools. Ecore is an object-oriented meta-model and it includes a graphical Eclipse-based editor for creating class models and generating Java code. While classes represent framework-provided concepts, annotations on classes and their properties are used to specify mapping definitions. Figure 7.2 shows a UML class diagram for part of the NUOPC FSML rooted at the feature `NUOPCModel`. Features are represented as classes or attributes and subfeatures are represented as containment references. Figure 7.3 shows an instantiation of classes (a FSM) and mappings to source code.

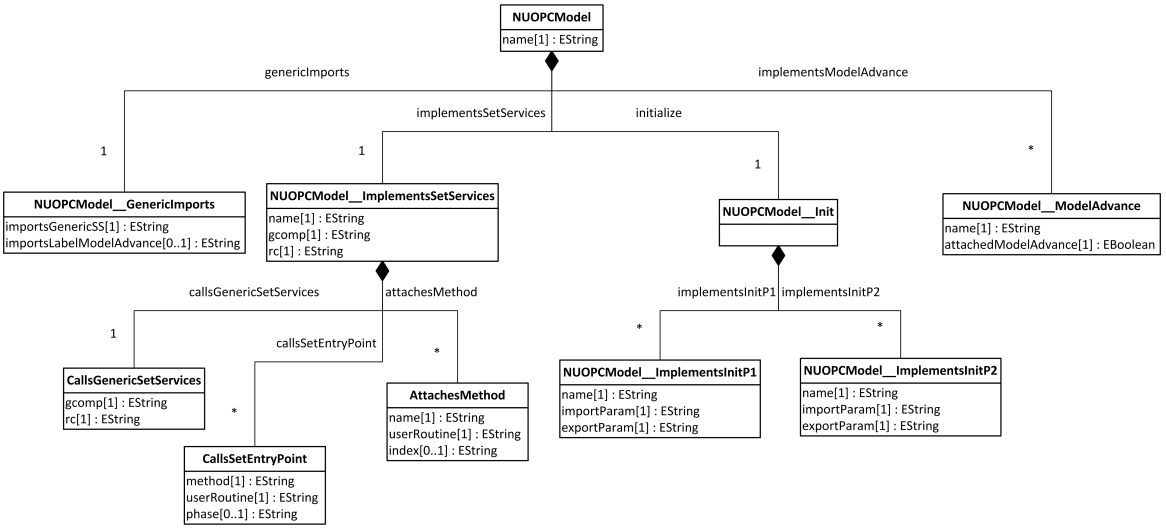


Figure 7.2: Partial NUOPC FSML class diagram rooted at `NUOPCModel`.

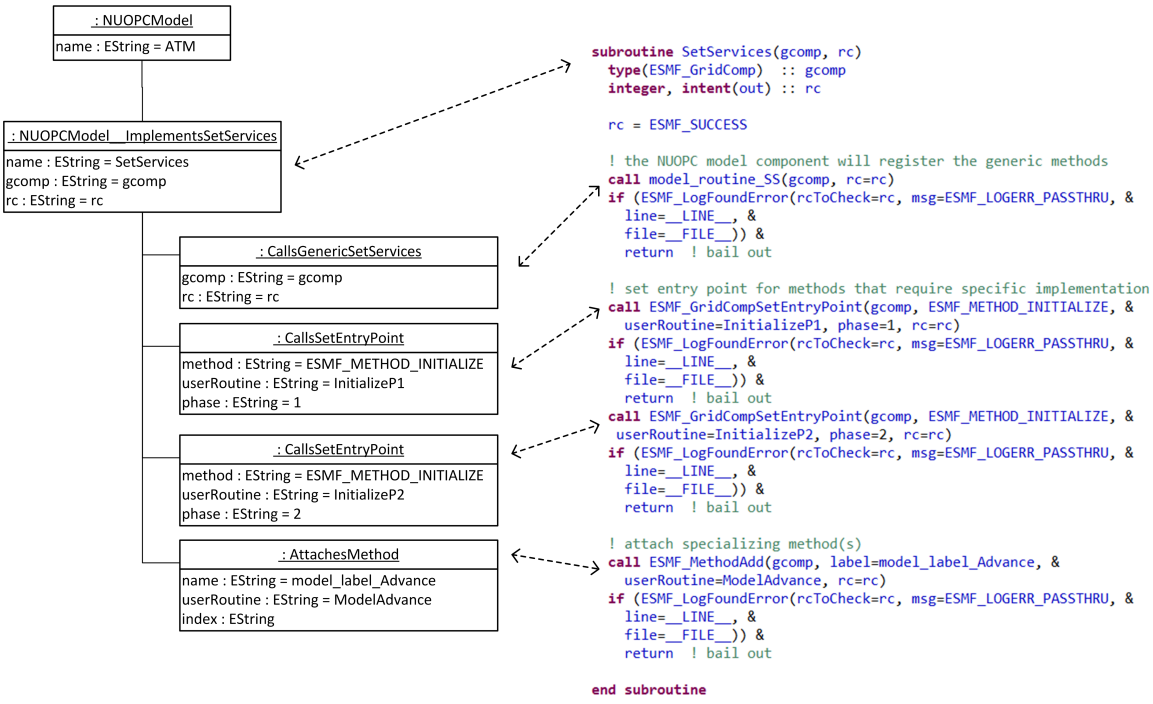


Figure 7.3: The object diagram on the left shows an instantiation of part of the NUOPC FSML corresponding to a `Model SetServices` subroutine. The dashed lines show mappings from FSM objects to source code.

Bibliography

[1] Micha? Antkiewicz and Krzysztof Czarnecki. Framework-specific modeling languages with round-trip engineering. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 692–706. Springer Berlin Heidelberg, 2006.