REPAST SIMPHONY BATCH RUNS GETTING STARTED

NICK COLLIER, JONATHAN OZIK - REPAST DEVELOPMENT TEAM

0. Before We Get Started

Before we can do anything with Repast Simphony, we need to make sure that we have a proper installation of the latest version. See the Repast Requirements Web Page for instructions on downloading and installing Repast Simphony and Java.[1]

1. Overview

A batch run in Repast Simphony consists of multiple individual model runs each using their own combination of parameters. The user defines a parameter space and uses the batch run capability to iterate through all the parameter combinations in that space. The number of individual runs is equal to the number of parameter combinations. Repast Simphony's batch run functionality iterates through the parameter space combinations and performs a run using each combination. These runs can be performed in parallel on the local machine, on remote machines, in the cloud or on a combination of the three. In addition runs can be done on a dedicated cluster. We will explore all of these options in the following sections.

The actual runs themselves are executed via a master client type architecture. A master process spawns child processes which perform the actual runs. The master process periodically polls these client processes to see if they have finished. Once they are finished the master process gathers the output from each client process and concatenates it together to form the total model output for all the runs. Each client works on its own subsection of the total number of batch parameter combinations, performing individual runs for each subsection. In this way, the individual runs are performed in parallel.

We will be using the JZombies demonstration model as an example. This a relatively simple model involving zombie agents chasing human agents and human agents running away from the zombies. The JZombies model can be imported together with the other demonstration models by starting Repast Simphony and choosing File → Import Repast Examples. If you click on the small downward facing triangle next to the Eclipse launcher button (fig. 1), you'll see the various available launchers. Click on "JZombies_Demo Model" to launch the demo model. You may have more entries than the single one shown in the screenshot, in which case, just be sure to select JZombies_Demo Model. Do not select Zombies_Demo Model.
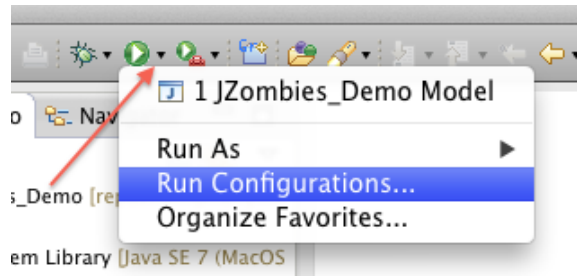

[1] https://repast.github.io/requirements.html

FIGURE 1. JZombies Model Launcher

2. THE BATCH RUN CONFIGURATION GUI

The Batch Run Configuration GUI is used to configure and launch the batch run. To start the configuration GUI, start the JZombies demo model (see section 1) and click on the lightning bolt in the toolbar (fig. 2). Note that you can also start the GUI as a stand-alone application. See section 6.4 for the details.

FIGURE 2. Batch Run Configuration GUI Launch

The GUI is used to configure a batch run. This configuration can be saved and then loaded at a future time. The configuration GUI consists of a tool bar and four tabbed panels (fig. 3). The first four toolbar buttons are used to create a new configuration, open an existing configuration, save the current configuration, and save the current configuration in a new file. Hovering over the buttons with the mouse will show a tooltip explaining the buttons function. The fifth button will reload the properties in the model panel from the current model.

The remaining two buttons have to do with performing the batch runs itself. Regardless of type of batch run (local, distributed, and so forth), the batch run requires enough of Repast Simphony as well as the model to be run in order to work. These parts are bundled into a "payload" archive file that is then run locally or transferred to another machine depending on whether the runs are local or to be distributed among remote machines. The
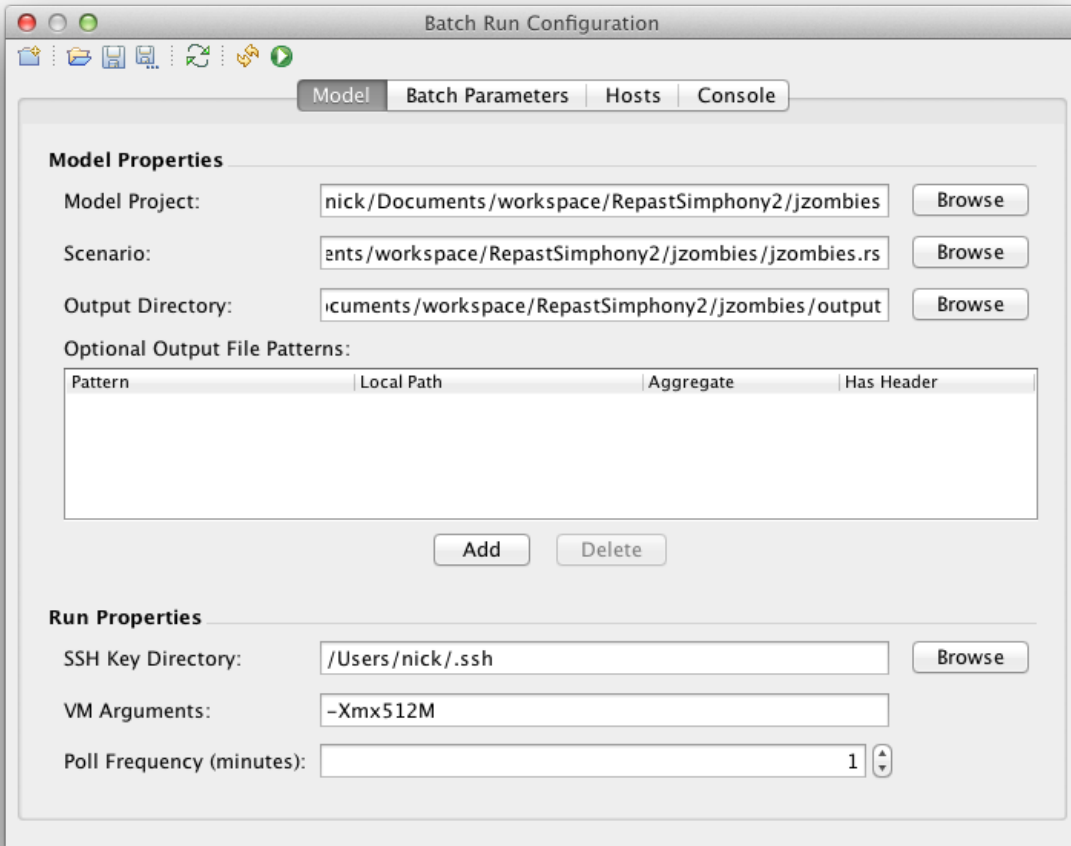
FIGURE 3. Batch Run Configuration GUI

second to last button ⚙ will create this payload archive file for the current configuration. The last button ⊙ will then execute the batch run.

2.1. **Model Panel.** When the batch configuration GUI is started and you have a model loaded in the Repast Simphony runtime then the GUI will create a new configuration with that model loaded. The model panel will contain various required model properties:

Model Project: The location of the project that contains the model.

Scenario: The location of the scenario directory of the model.

Output Directory: The directory where the model batch run output will be written.

The browse button next to each of these items can be used to select the directory for each property. The reload button ⟳ will load best guess defaults for these properties when the GUI is launched with an associated project.

By default the batch run mechanism will collect and concatenate any output files specified in the simphony model as File Sinks. If you do your own custom output that is not defined in a File Sink, you can use the "Optional Output File Patterns" table to define how to deal with such custom output. You provide a file pattern that can be used to identify the different instances of the custom output and how to treat that output once it is found. If the output is to be aggregated it will be concatenated and copied to the master machine. If its not aggregated, the output will still be copied to the master machine so you do not need to grab the files individually.

The fields in the table are defined as follows:

Pattern: The pattern specifies where the custom output is located. Unix "glob" style wildcards can be used, such as the *. For example, a pattern of "output/MyCustomOutput*.csv" means any files that begin with "MyCustomOutput" and end with ".csv" in an "output" directory will be collected as output and transferred back to the master machine.

Local Path: A directory into which the custom model output will be copied. This directory will be created beneath the Output Directory mentioned above.

Aggregate: If this is checked then the files that match the pattern specified in the Pattern field will be aggregated together into a single file.

Has Header: If this is checked then the first line in any files that match the pattern specified in the Pattern field will be treated as a Header and will not be aggregated as part of the data.

Specifying output patterns is optional, and only necessary if you do not generate file output using the default File Sink mechanism.

The Run Properties sections defines properties that are used during the actual model run.

SSH Key Directory: The directory that contains the user's public ssh rsa key. See the distributed runs section (section 2.3.2) below for more details.

VM Arguments: The virtual machine arguments to use when performing the actual model runs.

Poll Frequency: The frequency with which the master process will poll its clients to determine if they have finished.

The VM arguments should be space separated and quoted if the argument itself contains a space.

2.2. **Batch Parameters Panel.** The batch parameters panel (fig. 4) is used to define the parameter space that the batch run will iterate over.

It has two file properties.

FIGURE 4. Batch Parameters Panel

Parameter File: The location of the non-batch parameter file for the model. This is typically the parameters.xml file in the scenario directory.
Batch Parameter File: The location of the batch parameter file. By default, this will be the batch_params.xml file the model's batch directory.

The GUI will use the parameters file to generate some sample batch parameters if the batch parameters file is empty. The parameters file is also used during the actual runs and thus the parameters in the non-batch parameters file must match those in the batch parameters file. Beneath the batch parameter file text box, you will see the actual parameters themselves. If the batch parameter file is empty, then the GUI will display the parameters in the parameter file as constant values. The drop down combo box underneath each parameter name can be used to define the value or range of values for that parameter during a batch run. The possible parameter types are:

Constant: The parameter will be set to the specified value in the "value" box and this value will remain constant over all the batch runs.

Number Range: The parameter will begin at the value specified in the "From" text box and be incremented by the "Step" amount up to and including the "To" amount.

Space Separated List: The parameter will be set to each individual value in the "Values" list. The values should be space separated.

Random: Sets the value at random. This only applies to the Random Seed which will then be initialized with the current time.

The full combination of the parameters defines the parameter space. The batch runs will iterate over the entire space. For example, if the Random Seed is set to a constant of 1, the Human Count to a Number Range where From: 1, To: 3, Step: 1 and the Zombie Count is set to a Space Separated List of "20 15", then the parameters for each run will be:

(1) : Random Seed: 1, Human Count: 1, Zombie Count: 20
(2) : Random Seed: 1, Human Count: 2, Zombie Count: 20
(3) : Random Seed: 1, Human Count: 3, Zombie Count: 20
(4) : Random Seed: 1, Human Count: 1, Zombie Count: 15
(5) : Random Seed: 1, Human Count: 2, Zombie Count: 15
(6) : Random Seed: 1, Human Count: 3, Zombie Count: 15

Note that order of the runs with respect to the parameters may actually differ, but all the parameter combinations will be executed.

When the batch parameters as recorded in the batch parameter do not match the ones specified in the GUI, then the batch parameter file will be displayed in red. (This will occur when a parameter type or value is changed.) The red indicates that the batch parameter file needs to be re-generated using the "Generate" button in the lower right of the panel.

2.3. **Hosts Panel.** The Hosts panel (fig. 5) is used to define the machines on which the batch runs will be performed. Both local and remote hosts can be defined. The left hand side of the panel lists the hosts that been previously defined (e.g. "nick@localhost") and the right hand side "Host Properties" displays the properties for the host selected on the left hand side. New hosts can be created using the "Add Host" ⬜ button. A copy of an existing host can be made with the "Copy Host" 📄 button. Existing hosts can be deleted by selecting the host to delete and pressing the "Delete Host" ✖ button.

The Host Properties are:

Type: The type of host: local or remote.

User: The user name of the account on the remote machine. This is only used for a remote host.

Host: The name or ip address of the remote machine. This is only used for a remote host.

SSH Key File: The name of the private SSH key file used to auto-login on the remote host. This is only used for a remote host.
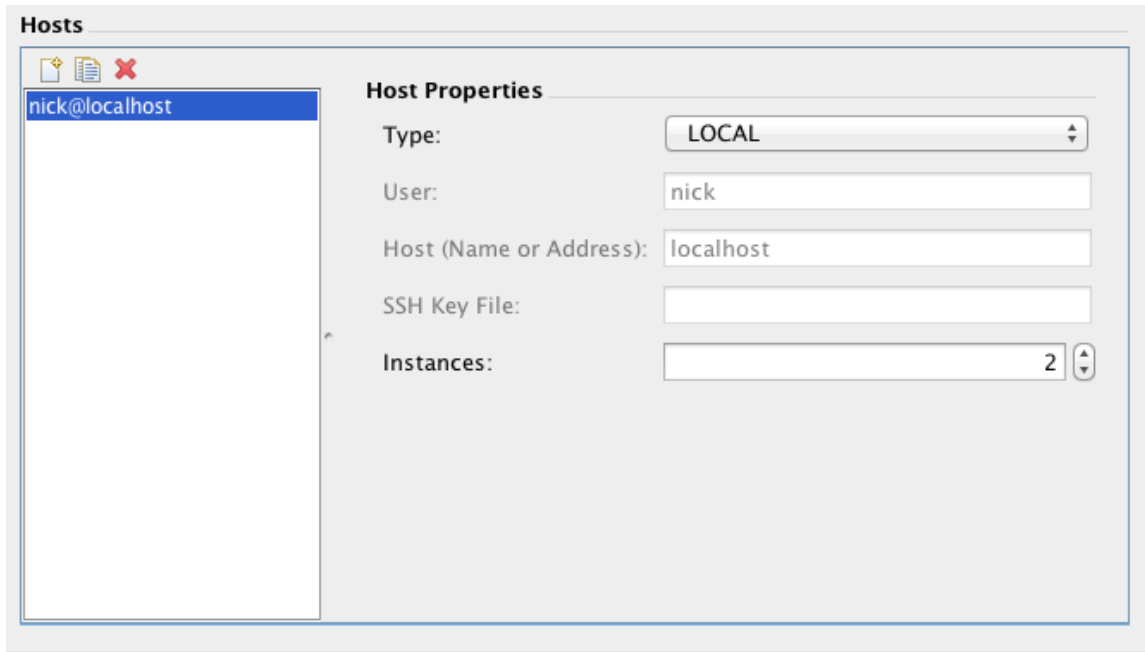
FIGURE 5. Hosts Panel

Instances: The number of instances to run on the host. Each instance will be assigned a chunk of the parameter space to run.

Each of the hosts defined on the host panel will be used to perform the batch runs. For example, if one local host and one remote host is defined and the local host has two instances and the remote host has a single instance, then the batch parameter space will be divided in thirds. Two-thirds will be run locally in parallel: each instance on the local machine will run a third of the space. The remaining third will be run on the remote machine. The number of instances to run on a machine should be roughly the number of hardware threads available on the machine. Of course, your milage may vary, especially if the machine is performing other computationally expensive tasks.

2.3.1. *Local Runs.* Local runs are the easiest runs to perform, requiring the least amount of external setup. To configure a local run, create a host (or chose an existing one) and set its type to LOCAL. You will see the host labeled as something like X@localhost where X is the user name of the account running the GUI. When you perform a local run, the GUI copies the archive payload mentioned above to a temporary location on the machine's hard-drive. This payload is unzipped and the parameter space is divided among the instances to be run locally and the runs are begun. Once the runs have ended, the output generated by

each instance is concatenated together and placed in the output directory specified in the model panel.

Note that if the only Java version you have installed on your machine is that included with Repast Simphony, then the runs may fail because a Java runtime cannot be found. If that is the case Java 8 and Java 11 JDKs can be downloaed from from Adopt Open JDK. Select either OpenJDK 8 with Hotspot or OpenJDK 11 with Hotspot. See https://stackoverflow.com/questions/52511778/how-to-install-openjdk-11-on-windows for help on installing under Windows and untar into /Library/Java/JavaVirtualMachines on macOS.

2.3.2. *Remote Runs.* Remote runs are more complicated and you may have to do some setup on the local and remote machine. Remote machines can be anything from miscellaneous spare machines that you might have up to amazon's cloud computing services. Remote runs work over Secure Shell (SSH). SSH is a cryptographic network protocol for secure data communication, remote shell services or command execution and other secure network services between two networked computers that connect, via a secure channel over an insecure network, a server and a client (running SSH server and SSH client programs, respectively). Any machine that allows you to connect to it via SSH and can run a java virtual machine can be used to perform a remote run. See section 6 on how to set up the remote machines.

On the local side, that is, on the machine you are running the batch configuration GUI on, you will need to be setup to auto-login to the remote machine using public key authentication. This consists of creating a ssh public / private key pair on your machine, if those don't already exist, copying that key to the remote machine and then appending the key to the list of authorized keys on the remote machine. Some useful references that describe this are:

- OSX - http://smbjorklund.no/ssh-login-without-password-using-os-x
- Windows - See below.
- Linux - https://hkn.eecs.berkeley.edu/∼dhsu/ssh_public_key_howto.html

Other sites exist as well, google "ssh public key authentication" for more information.

Currently, the new OpenSSH format is not supported (OpenSSH 7.8 and newer). Ensure the key you are using starts with `-----BEGIN RSA PRIVATE KEY-----` rather than `-----BEGIN OPENSSH PRIVATE KEY-----`.

The situation on windows is more complicated. OSX and Linux come with ssh and thus the built-in ability to generate public / private key pairs. Unfortunately, this is not the case for Windows and thus some setup is required. The following steps describe one way to generate the key pair and copy it to the remote machine under Windows.

(1) Download and install the Git Client from http://git-scm.com/download/win.
(2) From the start menu, run "Git Bash". You may need to navigate to the Git folder in "All Programs". Git Bash is terminal program in which you can type the commands to create the key pairs.

(3) Check if you have a key pair already generated. If the files id_rsa and id_rsa.pub exist in an .ssh directory in your home directory (e.g. C:\Users\nick\.ssh) then the keys already exist and you can skip to step 5.

(4) In the Git Bash terminal type `ssh-keygen -m PEM -C "your-email@example.com"`. We want the default settings so when asked to enter a file in which to save the key just press enter.[2] Do not specify a passphrase. You should see see something like figure 6

(5) In the Git Bash terminal type `scp ~/.ssh/id_rsa.pub user@machine:~` where user is the user name used to login to the remote machine and machine is the machine name or ip address. This will copy the public key into the home directory on the remote machine.

(6) Login to the remote machine. Type `ssh user@machine` replacing user and machine as appropriate. You will be asked for the password to login to the remote machine.

(7) Type `cat id_rsa.pub >> .ssh/authorized_keys` to add the public key to the list of authorized keys on the remote machine. If you get a "no such file or directory" message, type `mkdir .ssh; touch .ssh/authorized_keys` and then `cat id_rsa.pub >> .ssh/authorized_keys`. If you get an error that .ssh already exists when trying to create the directory with `mkdir`, type `touch .ssh/authorized_keys` and then `cat id_rsa.pub >> .ssh/authorized_keys`.

(8) Type `exit` to logout from the remote machine.

(9) Try to login to the remote machine again by typing `ssh user@machine` replacing user and machine. If you are NOT asked for your password but are asked for your ssh private key passphrase then the setup was successful.


```
Nicholson Collier@WIN-21EV8PRJUIV ~
$ ssh-keygen -t rsa -C "srcnick@sf.net"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Nicholson Collier/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Nicholson Collier/.ssh/id_rsa.
Your public key has been saved in /c/Users/Nicholson Collier/.ssh/id_rsa.pub.
The key fingerprint is:
53:03:b0:0d:7e:53:30:c5:ac:2d:84:98:e2:e5:85:5c srcnick@sf.net
```

FIGURE 6. Key Generation with Git Bash

Once you've got your keys on the remote machine, you can run remote batch runs using the Batch Run GUI. The host properties User, Host and SSH Key are used to login to the remote host. The User property should be the user name for the ssh login. The host is the

---

[2]Do specify a different name of the newly generated SSH key if you currently have one installed, as the default settings with overwrite `/.ssh/id_rsa` and `/.ssh/id_rsa`

name or ip address of the machine to login into and the SSH Key File is the private key file on the local machine used to auto-login to the remote machine. This file is typically called id_rsa in a .ssh directory in the user's home directory, but it may be different.

When performing a batch run using a remote host, the active payload will be copied to the account of the user specified in the host properties. After that, the run occurs as it does in the local version. The archive is unzipped and the parameter space is divided among the instances to be run on that remote host. When the instance runs have ended the output is copied back to the machine where the GUI was run, concatenated and written to the output directory. Note that the remote model runs will be performed using whatever Java runtime is the default on the remote machine. If you receive class format or other errors that indicate a Java runtime incompatibility, make sure that that the remote Java is compatible with the Java version with which you have compiled your code. Typically, code compiled with a later version of Java is not compatible with a previous version's runtime.

3. Preparing Your Model for Batch Runs

There are three requirements for a model to be successfully run in batch mode.

The first requirement is that your model must terminate each run. When running the model via the GUI this is typically done by clicking the stop button. In the batch run case, you must set and check some termination condition and then stop the model if that condition is met. Or, set the model to terminate at some specific tick. In the JZombies case, the first of these might look like:

```
if (context.getObjects(Human.class).size() == 0) {
    RunEnvironment.getInstance().endRun();
}
```

This checks to see if there are any more Humans and if not then it calls RunEnvironment to end the run. Context here is the model's master context. Something like this could be scheduled to execute once every tick either manually or via a scheduled method on the context builder.

The second option looks like:

```
RunEnvironment.getInstance().endAt(30);
```

This schedules the simulation to end at tick 30. It can be placed in the ContextBuilder for your model.

The second requirement concerns the model's file output. The batch mechanism assumes that the output will be written somewhere in the "instance" directory where the model runs. The best way to insure that this occurs is to specify the file with only the file name or a relative directory path followed by the file name. Make sure that the path or file name does not begin with a "/" (under OSX, Linux) or "X:\" or "\" (under windows where X is any drive letter). The name itself can be anything. Figure 7 is an example of an acceptable file. Output created via the Simphony's File Sinks will be automatically collected and aggregated. If you have created custom output, then you need to use the Output File Patterns GUI in the Batch Run Configuration GUI's model tab to specify where to find and how to treat that output.

The third requirement is the exclusive use of the `data` and `lib` project directories for model data and additional model libraries, respectively. The contents of these folders will be included in the "payload" archive file. Any data that is needed by the model will need to be referenced through the project's `data` folder, e.g., `"./data/input.csv"`. Any additional jar files required by the model that are in the `lib` directory will automatically be made available for the model at runtime.

FIGURE 7. Relative Path in File Sink Editor

4. BATCH RUN MODEL OUTPUT

As mentioned above the output from each instance will be concatenated and copied into the output directory specified in the GUI. The file sink output will look the same as a normal non-batch run except that the run numbers will likely not be in numeric order given that the concatenation makes no account for order. For example, the output from run 11 may appear before that from run 2.

```
"run","tick","Human Count","Zombie Count"
1,0.0,200,5
1,1.0,200,5
...
11,0.0,220,15
```

```
11,1.0,220,15
11,2.0,209,26
...
2, 0.0, 12, 200
...
```

For each file sink output there will be a batch param map text file that specifies the batch parameters that were used to initialize that run. For example,

```
"run","randomSeed","human_count","zombie_count"
1,1,200,5
11,1,220,15
2,1,250,5
```

Here run 1 was initialized with a random seed of 1, a human count of 200 and zombie count of 5. Run 11 was initialized with a random seed of 1 a human count of 220 and zombie count of 15.

You will also see a status_output.properties file in the output directory. This reports the final status of each local and remote instance. For example,

```
ec2-107-20-23-221.compute-1.amazonaws.com.ubuntu.1 = OK
ec2-107-20-23-221.compute-1.amazonaws.com.ubuntu.2 = OK
localhost.nick.1 = OK
```
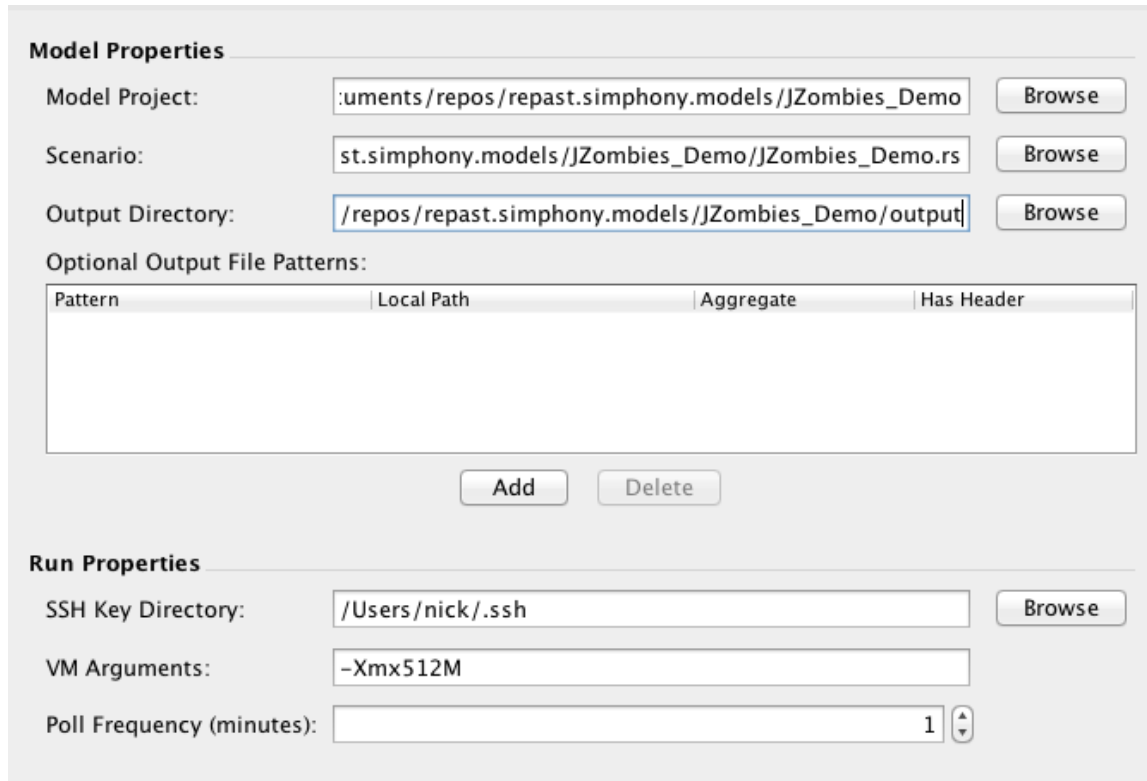
The format here is "hostname.username.instance number = status". In this case, we ran two instances on an amazon cloud machine with the name of ec2-107-20-23-221.compute-1.amazonaws.com under the ubuntu user. Both those finished successfully. We also ran one instance on the localhost under the nick user. That finished successfully as well. When the runs don't finish successfully you will see status other than "OK", such as "FAIL". There will also be additional log files in the output directory containing some details on the failure.

5. An Example

This section will walk you through an example batch run using a combination of local and remote machines. We will run 1 instance locally, 2 instances on a remote OSX machine and 1 instance on an Amazon EC2 cloud instance. We have set up the remote OSX machine and the Amazon cloud host using the instructions in section 6. The JZombies demo will be our example model.

We've set up the Model, Batch Parameters and Hosts Panel as in fig. 8, fig. 9 and fig. 10 respectively. Note that the IP address of the OSX machine as been obscured. If you want to follow along with the example, fill the Run Properties part of the Model panel as in the figure and set the batch parameters in the Batch Parameters panel to match the batch panel figure. For the hosts, create only a local host with two instances. To do that, go to the hosts panel, if there is an existing localhost entry, click on it and make sure the instances entry is 2. If there is no localhost entry, click the "Add Host" button and set its type to LOCAL and instances to 2.

FIGURE 8. Model Panel For Run

Once the various configuration properties have been set, we can click the run button. This will create the archive payload and start the runs. The Console panel will display information about the archive creation and then about the runs themselves. If you have a passphrase associated with your SSH key and you are doing remote runs you will be asked for your passphrase in the passphrase dialog (fig. 11).

The Console output reports the status of the batch run from setup through the actual runs. It can look daunting at first but it contains lots of useful information. Much of the output begins with a status keyword such as INFO, WARN or ERROR. If the run completes without any issues you should see only INFO type status messages. Following the status keyword is a timestamp followed by the name of the Java class that is producing the status message. After that, you will see the status message itself.

The first status message tells us that a "config.props" file has been written to the output directory. The config.props file contains all the batch run properties that were entered into the GUI. The code that performs the batch runs reads this config file and performs the runs

FIGURE 9. Batch Panel For Run

using that information. (In what follows, the file paths have been shortened for formatting purposes. You will see the full file path).

```
INFO  [AWT-EventQueue-0] 16:55:18,582
  repast.simphony.batch.gui.BatchConfigMediator - Writing batch
  run config file to: JZombies_Demo/output/config.props
```

The second status message says that the batch_params.xml file that was created via the GUI is "unrolled" and written to a temporary directory. "Unrolling" here refers to explicitly producing and writing each individual parameter combination to a file. It is this unrolled parameter file that is divided and distributed to the hosts for running.

```
INFO  [AWT-EventQueue-0] 16:55:18,588
  repast.simphony.batch.gui.BatchConfigMediator -
  Unrolling batch parameter file: JZombies_Demo/batch/batch_params.xml to
  /var/folders/31/yd0l22kx35zgy1wbnc8p7gbr0000gr/T/unrolledParamFile.txt
```

The next section is the output of the Ant script (see http://ant.apache.org) that creates the archive payload. Here we can see the creation of a working directory underneath a temporary directory. This working directory will contain the model itself and the Repast Simphony code necessary to run the model. The [copy] and [jar] sections report model code and files being copied into the working directory and the creation of jar archives
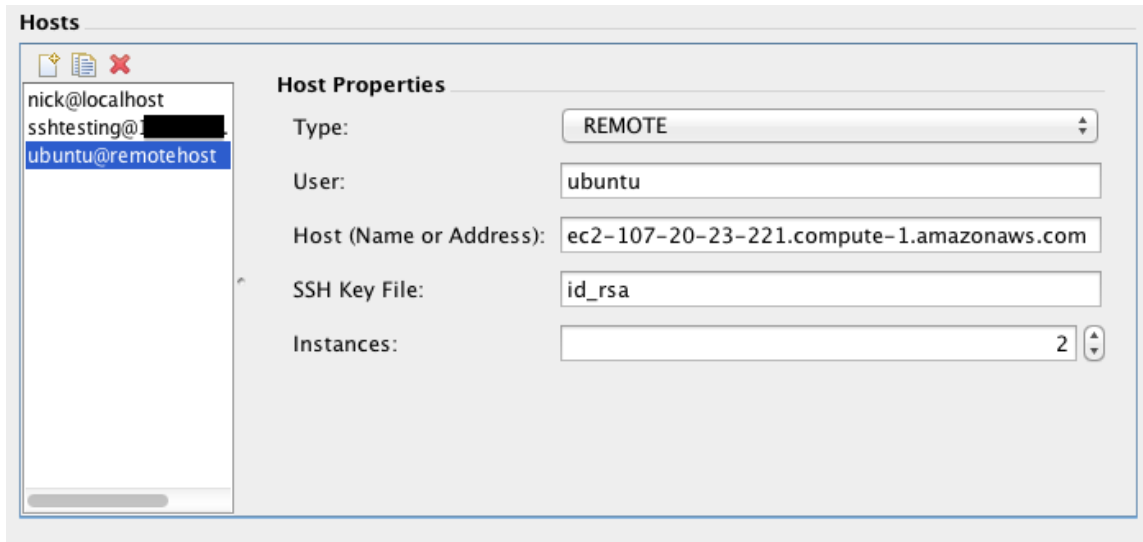
FIGURE 10. Hosts Panel For Run

containing the necessary repast code. (I've replaced part of the temporary path with "..." and truncated the jar section a bit to improve the readability here.)

```
make_zip:
    [delete] Deleting directory /private/var/folders/.../working
    [mkdir] Created dir: /private/var/folders/.../working
    [mkdir] Created dir: /private/var/folders/.../working/bin
    [mkdir] Created dir: /private/var/folders/.../working/scenario.rs
    [copy] Copying 2 files to /private/var/folders/.../working
    [copy] Copying 3 resources to /private/var/folders/.../working
    [copy] Copying 1 resource to /private/var/folders/.../working/bin

make_pre_reqs:
    [mkdir] Created dir: /private/var/folders/.../working/data
    [mkdir] Created dir: /private/var/folders/.../working/lib
    ...
    [jar] Building jar: /private/var/folders/.../working/lib/repast.simphony.batch.jar
    [jar] Building jar: /private/var/folders/.../working/lib/repast.simphony.statecharts.jar
    [copy] Copying 69 files to /private/var/folders/.../working/lib
    [jar] Building jar: /private/var/folders/.../working/lib/model.jar
    [copy] Copying 10 files to /private/var/folders/.../working/scenario.rs
    [move] Moving 1 file to /private/var/folders/.../working/scenario.rs
```

```
[jar] Building jar: JZombies_Demo/output/complete_model.jar
[echo] Completed building model archive.
```

The archive payload is then copied to the local and remote hosts. The process here is that a zip archive is created containing the archive payload as well as some additional configuration data. This zip archive is named with the user name followed by the host together with a timestamp (e.g. nick_localhost4106394375716993975.zip). In the case of a local host, the zip archive is copied to a local temporary directory whose name begins with simphony_model_ followed by a timestamp (e.g. simphony_model_1368219324898).

```
INFO  [SwingWorker-pool-1-thread-2] 16:56:26,990
  repast.simphony.batch.ssh.LocalSession -
  Copying locally /var/folders/.../nick_localhost4106394375716993975.zip
  to  /var/folders/.../simphony_model_1368219324898
INFO  [SwingWorker-pool-1-thread-2] 16:56:27,411
  repast.simphony.batch.ssh.LocalSession - Copying Finished.
```

The remote host works similarly. The zip archive is copied to a simphony_model_ plus timestamp directory on the remote host. In this case, the zip archive is named ubuntu_ec2-107-20-23-221.compute-1.amazonaws.com8807870753490178805.zip and the remote directory is /simphony_model_1368219324898/.

```
INFO  [SwingWorker-pool-1-thread-2] 16:56:09,829
  repast.simphony.batch.ssh.RemoteSession -
  Copying /var/folders/.../
  ubuntu_ec2-107-20-23-221.compute-1.amazonaws.com8807870753490178805.zip to
  ubuntu@ec2-107-20-23-221.compute-1.amazonaws.com:~/simphony_model_1368219324898/
  ubuntu_ec2-107-20-23-221.compute-1.amazonaws.com8807870753490178805.zip ...
INFO  [SwingWorker-pool-1-thread-2] 16:56:23,197
  repast.simphony.batch.ssh.RemoteSession - Copying Finished.
```

Before a remote run is attempted, a check is made to see if Java is installed on the remote host. If it is not, then you will see an error. In this case, we can see that the java version is 1.8.0_232.

```
INFO  [SwingWorker-pool-1-thread-2] 16:56:33,369
  repast.simphony.batch.ssh.RemoteSession -
  Checking for java on ubuntu@ec2-107-20-23-221.compute-1.amazonaws.com
INFO  [Connect thread ec2-107-20-23-221.compute-1.amazonaws.com session] 16:56:33,691
  repast.simphony.batch.ssh.SSHSession - java version "1.8.0_232"
INFO  [Connect thread ec2-107-20-23-221.compute-1.amazonaws.com session] 16:56:33,692
  repast.simphony.batch.ssh.SSHSession - OpenJDK Runtime Environment (IcedTea 2.3.9)
  (1.8.0u232-2.3.9-0ubuntu0.11.10.1)
INFO  [Connect thread ec2-107-20-23-221.compute-1.amazonaws.com session] 16:56:33,692
  repast.simphony.batch.ssh.SSHSession -
  OpenJDK 64-Bit Server VM (build 23.7-b01, mixed mode)
```

Once the java version has been checked and assuming it is OK, then the archive payload is unzipped and run.

```
INFO  [SwingWorker-pool-1-thread-2] 16:56:38,685
  repast.simphony.batch.ssh.LocalSession - Unzipping model
  /var/folders/.../simphony_model_1368219324898/nick_localhost4106394375716993975.zip
INFO  [SwingWorker-pool-1-thread-2] 16:56:39,177
  repast.simphony.batch.ssh.LocalSession - Running model on localhost ...


INFO  [SwingWorker-pool-1-thread-2] 16:56:34,583
  repast.simphony.batch.ssh.RemoteSession - Unzipping model on
  ubuntu@ec2-107-20-23-221.compute-1.amazonaws.com
INFO  [SwingWorker-pool-1-thread-2] 16:56:37,633
 repast.simphony.batch.ssh.RemoteSession - Running model on
 ubuntu@ec2-107-20-23-221.compute-1.amazonaws.com ...
```

The remote and local hosts are then periodically polled to see if they have finished. The polling status updates end with "DONE:" followed by a "yes" if the runs on that host have finished. Otherwise "no" will be output.

```
INFO  [pool-3-thread-3] 16:56:51,210 repast.simphony.batch.ssh.LocalDonePoller -
  Polled /var/folders/.../simphony_model_1368219324898 for DONE:  yes
INFO  [pool-3-thread-2] 16:56:52,540 repast.simphony.batch.ssh.RemoteDonePoller -
 Polled simphony_model_1368219324898 on ec2-107-20-23-221.compute-1.amazonaws.com
 for DONE: yes
INFO  [pool-3-thread-1] 16:56:52,796 repast.simphony.batch.ssh.RemoteDonePoller -
  Polled simphony_model_1368219324898 on XXX.XXX.XXX.XXX
  for DONE: yes
```

If there was error during the run, that should be reported here as part of the polling. Assuming there wasn't then the local and remote output is copied and concatenated into the output directory that was specified in the GUI.

```
INFO  [SwingWorker-pool-1-thread-2] 16:56:55,630
  repast.simphony.batch.ssh.RemoteSession - Finding and copying remote
  output from sshtesting@XXX.XXX.XXX.XXX:simphony_model_1368219324898
  to /var/folders/.../sshtesting_XXX.XXX.XXX.XXX
INFO  [SwingWorker-pool-1-thread-2] 16:56:58,412
  repast.simphony.batch.ssh.RemoteSession - Finding and copying remote
  output from ubuntu@ec2-107-20-23-221.compute-1.amazonaws.com:
  simphony_model_1368219324898 to /var/folders/.../
  ubuntu_ec2-107-20-23-221.compute-1.amazonaws.com
INFO  [SwingWorker-pool-1-thread-2] 16:57:00,247
  repast.simphony.batch.ssh.LocalSession - Finding output on localhost in
  /var/folders/.../simphony_model_1368219324898
INFO  [SwingWorker-pool-1-thread-2] 16:57:00,266
  repast.simphony.batch.ssh.SessionsDriver - Aggregating output into
```

```
  JZombies_Demo/output
INFO  [SwingWorker-pool-1-thread-2] 16:57:00,267
  repast.simphony.batch.ssh.SessionsDriver - Finished. Elapsed Time: 1.5906
```

If the Java version (local or remote) is greater than 11, you may see errors like:

```
INFO  [Thread-1] 16:57:38,496 repast.simphony.batch.ssh.LocalSession -
  ERROR [main] 16:57:38,494 repast.simphony.batch.InstanceRunner -
  Error while running model
INFO  [Thread-1] 16:57:38,496 repast.simphony.batch.ssh.LocalSession -
  repast.simphony.scenario.ScenarioLoadException: Unable to make protected
  final java.lang.Class java.lang.ClassLoader.defineClass(java.lang.String,byte[],
    int,int,java.security.ProtectionDomain) throws java.lang.ClassFormatError
    accessible: module java.base does not "opens java.lang" to unnamed
    module @324e4822
```

If possible, we recommend using Java/JRE 11 for the batch run Java. However, these errors can be fixed by using the `--add-opens` VM argument. The format is `--add-opens <module>/<package>=ALL-UNNAMED`. For example, if the error specifies `module java.base does not "opens java.util to unnamed module @371ef662`, then the module is `java.base` and the package is `java.util`, and the VM argument is `--add-opens java.base/java.util=ALL-UNNAMED`.

The `--add-opens` arguments can be added to the VM arguments text box in the model panel (2.1). The should be separated by spaces.

There are likely to be more than one of these arguments required. For example, the JZombies demo needs:

```
--add-opens java.base/java.lang=ALL-UNNAMED
--add-opens java.base/java.util=ALL-UNNAMED
--add-opens java.base/java.lang.reflect=ALL-UNNAMED
--add-opens java.base/java.text=ALL-UNNAMED
--add-opens java.desktop/java.awt.font=ALL-UNNAMED
```
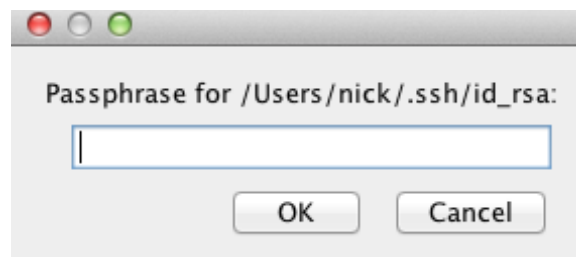

Figure 11. Passphrase Dialog

6. Remote Machine Setup

In order to perform a run on a remote machine that machine must be set up to accept an ssh connection and you must have an account on the remote machine. Once the remote machine has been setup for ssh logins, you then need to add your public key to its authorized_keys file as described in section 2.3.2. The remote machine must have at least Java 8 installed and it must be be the default Java on that machine.

6.1. OSX. Allowing remote logins on OSX is done using the System Preferences on the machine you want to remotely login to. Start System Preferences and double click on the sharing icon. Select the Remote Login as seen in figure 12. Make sure you allow access for the user you will be logging in as. You should now be able to ssh into the remote machine and copy your public keys to the server as described in links given in section 2.3.2.
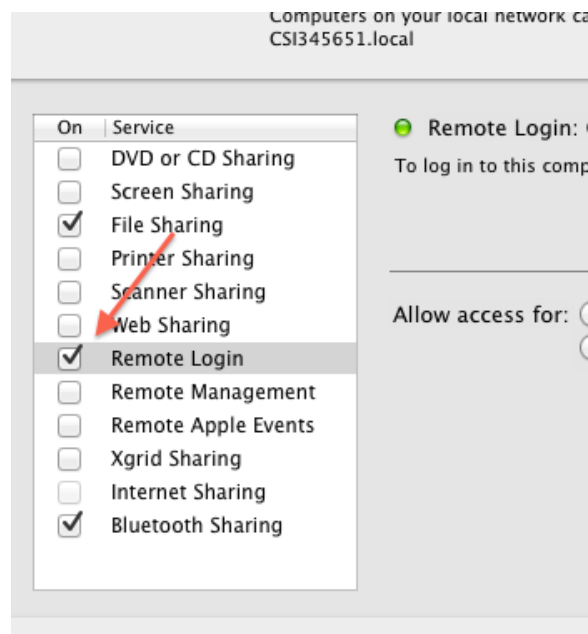

Figure 12. Enabling SSHD on OSX

6.2. Linux / Unix. Chances are if the remote machine is running Linux or another Unix variant it is already set up to allow ssh logins. If not, follow your distributions instructions for turning on the sshd service. You should now be able to ssh into the remote machine and copy your public keys to the server as described in section 2.3.2.

6.3. **Windows.** Windows does not come with a SSH server and so one must be provided. As a solution to this, we've provided a Virtual Box virtual machine running a linux distribution. You can run this on your windows machine and the remote batch runs will be performed on that. (Again, this is only required to use a windows machine as a remote machine not to do local runs on a windows machine.)

(1) Download and Install Virtual Box from https://www.virtualbox.org/wiki/Downloads.
(2) Download Repast Simphony virtual machine file (Repast_VM_Ubuntu_12.0.4.ova) from http://sourceforge.net/projects/repast/files/Repast Simphony/Repast Simphony 2.1/Repast_VM_Ubuntu_12.0.4.ova/download
(3) Double Click on the downloaded virtual machine. Virtual Box should start with the Import Virtual Appliance Dialog up (fig. 13).
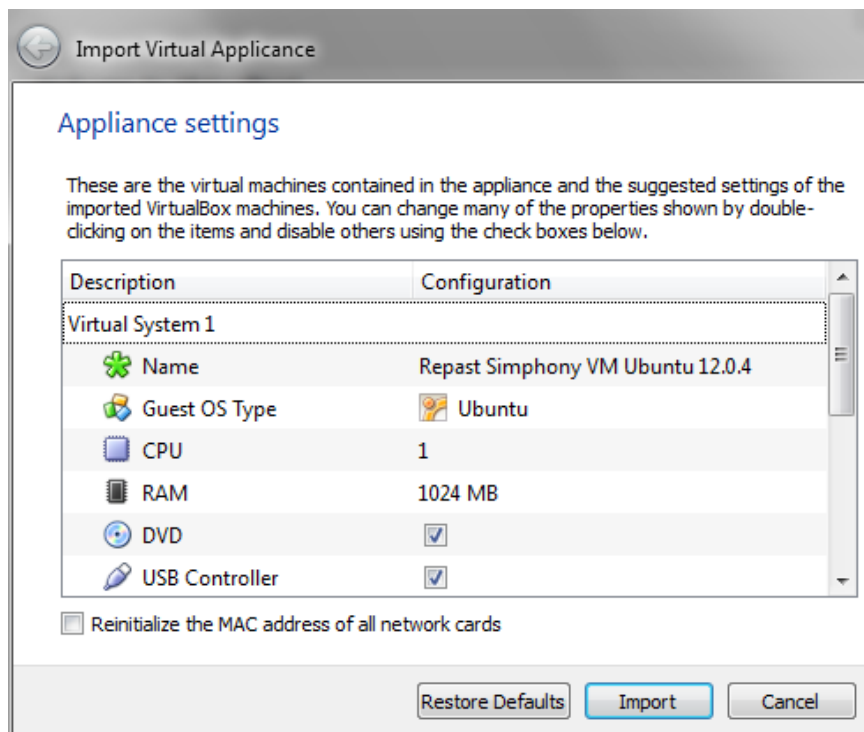(4) Click the Import button to import the virtual machine.

FIGURE 13. Import the Repast Simphony Server Virtual Machine

To run the virtual machine so that it can be used do the batch runs:

(1) Start Virtual Box

(2) Start the virtual machine. You should see it on the left hand side named Repast Simphony VM Ubuntu 12.0.4. (fig 14). Double click it. While the virtual machine is starting you will see some Virtual Box Information messages.
(3) If you see an error message about network settings, click the "Change Network Settings" button (fig. 15) and then the OK button in the settings dialog that pops up.
(4) The virtual machine will then run through its boot process (this may take a minute or two the first time) and eventually you will be presented with a command prompt (fig. 16). (The virtual machine is configured as a server without a GUI.)
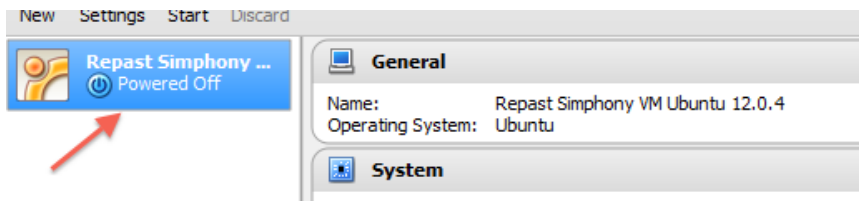

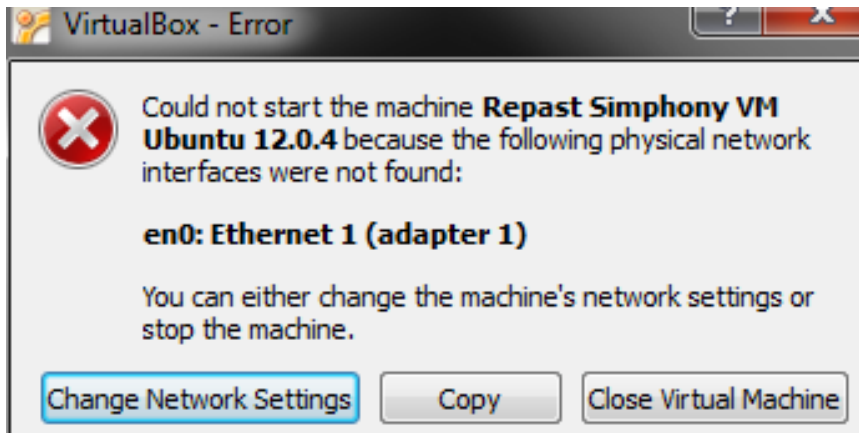FIGURE 14. Repast Simphony Server Virtual Machine

FIGURE 15. Network Error Dialog

At this point, the virtual machine is almost ready to do the batch runs. You still need to (1) get the IP address of the virtual machine in order to connect to it, and (2) you need to copy your public key over to it. To find the IP address:

FIGURE 16. VM Login

(1) Boot the virtual machine, if you haven't already.
(2) Login to the virtual machine using a username of `simphony` and a password of `simphony`.
(3) Once logged in, you should see a welcome message that includes the ip address. It should say something like: `IP Address for eth0:  XXX.XXX.XXX.XXX` where the Xs are replaced with your IP address (fig. 17). Note that eth0 might also be different. You can also get the IP address by running the ifconfig command. You will see the details of all the installed network interfaces. These include an "inet addr" entry that specifies the ip address.

FIGURE 17. IP Address

Once you have the IP address you can use that as the Host for batch runs. The user will again be `simphony`. Lastly, you will still have to perform the one time copy of your public key to this machine as described in the Remote Runs section (section 2.3.2). Once that has been accomplished, the machine is ready. Note that you do not need to log directly into it, it only needs to be running. The default password is `simphony`. If you want to change it, you can use the `passwd` command.

To shut down the virtual machine when you no longer need it, you can close the window as normal. This should pop up a Close Virtual Machine Dialog. Make sure the Send the shutdown signal is selected and then click OK.

6.4. **Amazon Cloud Services.** Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. It is possible to use this remote compute capacity to perform remote batch runs. More on Amazon's EC2 can be found at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

Amazon EC2 based batch runs begin with setting up a remote machine instance that runs an Amazon Machine Image (AMI). The batch run will login into this instance using SSH and perform the runs on it. The setup only needs to be done once. After that, the instance can be stopped and started as needed. The Amazon EC2 Getting Started docs describe this process: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html. (Note that Amazon, at the time of writing, has 12 month free tier. See the links in the EC2 Getting Started Guide for more info.) Follow the Getting Started Guide to set up your instance. Choose the 64 bit Ubuntu Server 18.04 LTS (HVM) (figure 18) as the Amazon Machine Image (AMI) when creating an instance.
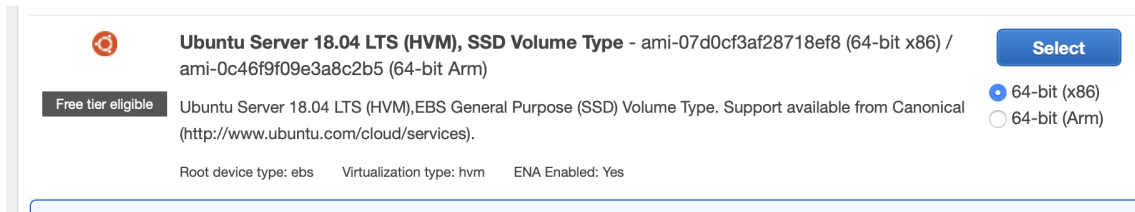

Figure 18. Select the Ubuntu Server 18.04 LTS AMI

Once your instance is running, log into and explore the instance using SSH as described in Step 2 of the Amazon Getting Started Guide. You will need to install Java on the instance. From the command line that you should see after you've logged in, run the following commands (type them in and press enter after each one):

```
sudo apt-get update
sudo apt-get install openjdk-11-jre-headless
sudo apt-get install zip
```

After each command, you may get a question about whether to continue with the install. Accept the default 'Yes' by pressing the enter key. Once you've done this, your instance should be ready to perform a batch run. The username to log in with is ubuntu as is the password. To see the address of your instance, click on the instance in your EC2 Instances console. The host address for your batch configuration should appear in the instance details below it (figure 19). Make sure to stop your instance when you are done using it. When you restart the instance the next time, the address will have changed so be sure to update your batch configuration properties.

Amazon EC2 has its own public key authentication setup that is described in its Getting Started document. Consequently, the procedures described in section 2.3.2 are not required here.
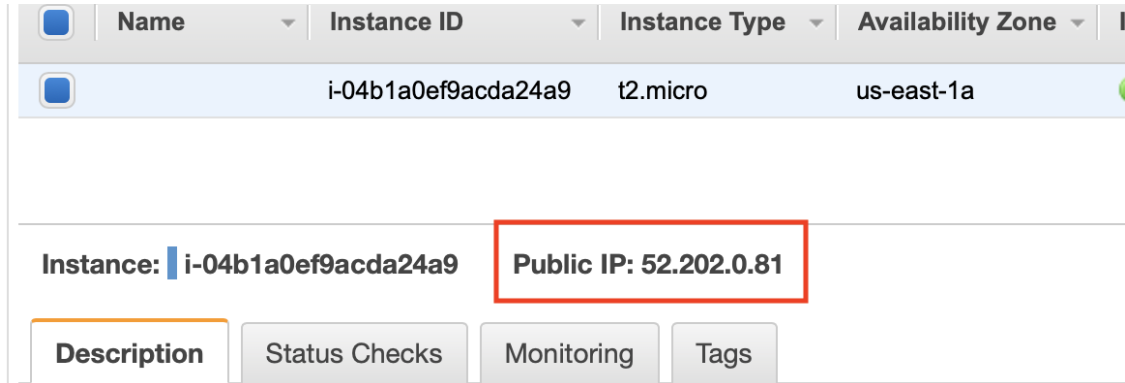

FIGURE 19. Instance Address

7. LAUNCHING THE BATCH RUN CONFIGURATION GUI

Batch Run Configuration GUI can be launched in 3 different ways.

(1) Using the Launch Button found in the runtime
(2) Using the Eclipse Launcher in your project's launchers folder
(3) Using the stand alone Batch Run applications

We have covered the first of these in section 2. The remaining two allow you to launch the configuration GUI and thus batch runs without launching the repast runtime.

7.1. Launching from Eclipse. To launch the batch GUI from the eclipse launcher, open the launchers folder in your model's eclipse project and right click on the launch file that begins with "Batch" (fig 20). Select "Run As" from the pop up menu and then the Batch model entry. This will launch the batch GUI and load the selected model into it. Note that when running via the launcher some of the output that would normally be in the Batch Run GUI Console will show up in Eclipse's console.

7.2. Launching as a Standalone Application. To launch the batch GUI as a stand alone application navigate to where Repast Simphony is installed. Double click the batch_runner.jar.
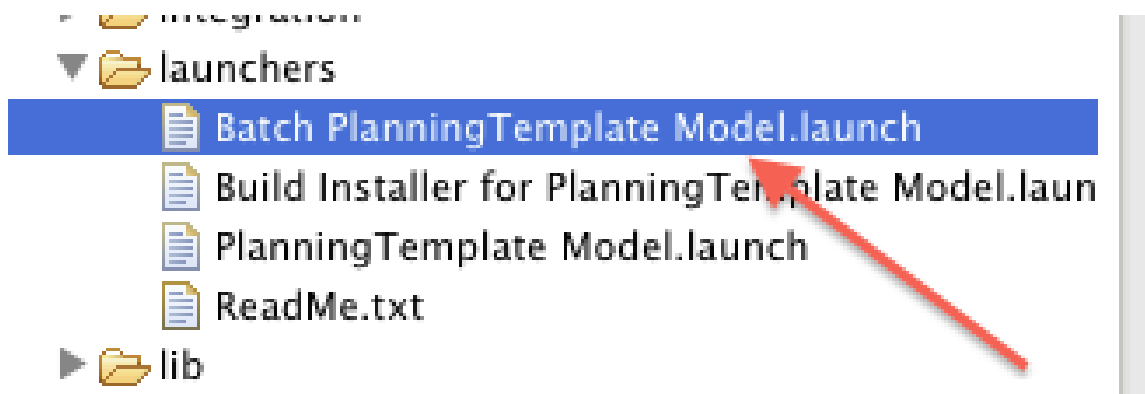
Figure 20. Eclipse Launcher for Batch Run GUI

8. Running on a Cluster or HPC Machine

It is also possible to perform batch runs on a cluster or a high performance computing (HPC) type machine. Such machine are composed of many processors each of which can run one or more batch run instances and all of which have access to a shared file system. Cluster / HPC runs will use the payload archive produced by the GUI but are not themselves run via the GUI. These batch runs will use the job submission paradigm that is standard on these types of machines.

The first step for running a model on an HPC machine is to copy the payload complete_model.jar to the remote machine. The payload is located in the model's output directory and can be created using the Batch Configuration GUI with the create archive button (⟳). The transfer is often done via scp or similar secure copy methods. After copying the payload to the desired location, the next step is to unzip the payload:

```
unzip complete_model.jar
```

Simphony includes submission files for two common job schedulers, PBS and Slurm. If your system uses a different scheduler, it should be possible to adapt instructions below.

8.1. **PBS.** We begin with the Portable Batch System (PBS) job scheduler. The first file to look at is the repast.pbs file. This is the file that is submitted to the PBS scheduler. It calls repastwrapper.sh to launch each instance and perform the actual runs. The top section of the file looks like:

```
#!/bin/bash
#PBS -N <Job_Name>
#PBS -q <Queue_Name>
#PBS -l nodes=<nodes>:ppn=<ppn>
#PBS -m e
```

where the elements in the angled brackets are meant to be edited for each individual use case. So, for example, if we wanted to name the job "Zombies_Test", submit the job to the "shared" queue, use 2 nodes and 8 processors per node we might issue the following `sed` command, all one line:

```
sed -i -e 's/<Job_Name>/Zombies_Test/' -e 's/<Queue_Name>/shared/'
-e 's/<nodes>/2/' -e 's/<ppn>/8/' repast.pbs
```

Alternatively the repast.pbs file can be edited by hand. The particulars for each system may be slightly different so you may need to consult with the reference manual or online help of the system you will be using.

The next step is to make sure that all the scripts are executable by issuing the command:

```
chmod +x *.sh
```

At this point the job is ready for submission and can be submitted from within the unzipped complete_model jar directory via the command:

```
qsub repast.pbs
```

The job is submitted to the specified queue and will run when the scheduler decides to run the job. Depending on the particular setup of the scheduler, you may receive an email alert when the job has finished running.

Whichever method you use to figure out when the job has completed executing, you can run an output combiner to combine the data output from the individual simulations that were run with the command:

```
./outputcombiner.sh
```

The combines data will be placed in a combined_data directory in the current directory.

8.2. **Slurm.** The slurm submission script is repast.slurm. It calls repastwrapper_slurm.sh to launch each instance and perform the actual runs. The basic idea is that we calculate the total number of process available for work, and run an instance on each process. The number of individual runs for each instance is determined by dividing the total number of runs (the number of lines in the unrolled parameter file) by the number of processes and distributing any remainder across instances as appropriate. The submission script (repast.slurm) defines the total number of processes to allocate by specifying the number of nodes and the number of tasks per node. The wrapper (repastwrapper_slurm.sh) does the actual math to determine which lines of the full unrolled parameter file that a particular instance will run, using the unique process id (SLURM_PROCID) for each instance.

The first step is specify the required options in ther repast.slurm file. The top section of the file looks like:

```
#!/bin/bash
#SBATCH --job-name=<Job_Name>
#SBATCH --partition=<Queue_Name>
#SBATCH --nodes=<nodes>
#SBATCH --ntasks-per-node=<ppn>
#SBATCH --time=<walltime>
```

```
#SBATCH --output=output.txt
#SBATCH --mail-user=<email>
#SBATCH --mail-type=all
```

where, as in the PBS case, the elements in the angled brackets are meant to be edited for each individual use case. So, for example, if we wanted to name the job "Zombies_Test", submit the job to the "shared" queue, use 2 nodes and 8 processors per node with a wall time of 1 hour an email of klopp@lfc.com we might issue the following `sed` command, all one line:

```
sed -i -e ’s/<Job_Name>/Zombies_Test/’ -e ’s/<Queue_Name>/shared/’
-e ’s/<nodes>/2/’ -e ’s/<ppn>/8/’ -e ’s/<walltime>/01:00:00/’
-e ’s/<email>/klopp@lfc.com/’ repast.slurm
```

Alternatively the repast.slurm file can be edited by hand. The particulars for each system may be slightly different so you may need to consult with the reference manual or online help of the system you will be using. You can always add any additional arguments to the SBATCH or srun arguments, of course. For example, you may have to set mem-per-cpu if the default on your hpc machine is not adequate.

The next step is to make sure that all the scripts are executable by issuing the command:

```
chmod +x *.sh
```

At this point the job is ready for submission and can be submitted from within the unzipped complete_model jar directory via the command:

```
sbatch repast.slurm
```

The job is submitted to the specified queue and will run when the scheduler decides to run the job. Depending on the particular setup of the scheduler, you may receive an email alert when the job has finished running.

Whichever method you use to figure out when the job has completed executing, you can run an output combiner to combine the data output from the individual simulations that were run with the command:

```
./outputcombiner.sh
```

The combines data will be placed in a combined_data directory in the current directory.

9. Command Line Batch Run Capabilities

The `batch_runner.jar` that comes with Repast Simphony provides a command line interface for launching the Batch Configuration GUI, for launching batch runs directly and for creating `complete_model.jar` payloads.

9.1. **Launching the Batch Configuration GUI.** The regular Batch Configuration GUI can be launched from the command line with the command:

```
java -jar batch_runner.jar
```

9.2. **Headless Options.** When supplying the `-hl` or `--headless` options, the command line utility will not launch the Batch Configuration GUI. Rather, it provides a command line interface to either directly execute batch runs or to create `complete_model.jar` payloads.

9.2.1. *Command Line Launching of Batch Runs.* The `-r` or `--run` options allow for direct launching of Batch runs from the command line. The easiest way to provide the necessary configuration information to the Batch system is to additionally specify, using the `-c` or `--batch_config` options, a configuration file. The following is an example of a configuration file:

```
# sample_batch_configuration.properties
model.directory=/path/to/my_model
scenario.directory=/path/to/my_model/my_model.rs
param.file=/path/to/my_model/my_model.rs/parameters.xml
batch.param.file=/path/to/my_model/batch/batch_params.xml
output.directory=model_outputs

output.pattern.1.pattern=hello_*.txt
output.pattern.1.path=hello_folder
output.pattern.1.concatenate=false
output.pattern.1.header=false

output.pattern.2.pattern=custom_output.csv
output.pattern.2.path=combined_custom_output.csv
output.pattern.2.header=true
output.pattern.2.concatenate=true

vm.arguments=-Xmx512M
key.directory=/user/home/.ssh
poll.frequency=0.1
host.0.type=LOCAL
host.0.instances=3
```

The options specified are the same as those defined within the Batch Configuration GUI[3]. With a batch configuration file in place, a headless batch run can be launched via:

```
java -jar batch_runner.jar -hl -r -c sample_batch_configuration.properties
```

Properties within the configuration file can be overridden by specifying `-model_dir` for the model directory, `-o` or `--output_dir` for the output directory, `-s` or `--scenario_dir` for the scenario directory, `-p` or `--params` for the parameters file and `-b` or `--batch_params` for the batch parameters file.

9.2.2. *Create Batch Payloads.* When in headless mode, if the `-r` or `--run` options are not specified, only a `complete_model.jar` payload is created in a specified output directory and the model is not run. This is helpful for automated generation of payloads that are

---

[3] When the "save" button is clicked within the Batch Configuration GUI, a properties file with this content is saved.

later run on remote resources, e.g., as seen in Sec. 8 or when using the Extreme-scale Model Exploration with Swift (EMEWS) framework [4].

In addition, as an alternative to a parameter sweep specified by a batch parameter file, it is sometimes helpful to provide a custom unrolled parameter file to the payload. The unrolled parameter file, discussed in Sec. 5, is a text file that contains each individual parameter combination as a separate line. The format for each line is the following:

```
run_number<tab>param1<tab>val1,param2<tab>val2,...,paramN<tab>valN
```

where `param1,...,paramN` correspond to the full set of parameters specified in the model's `parameters.xml` file[5]. Once an unrolled parameter file is created, it can be specified on the command line using the `-u` or `--upf` options. When the `-u,--upf` option is used only the creation of the payload will take place and the `-r,--run` option cannot be combined with it.

9.3. **Command Line Help.** For convenience, the command line utility help menu can be accessed via:

```
java -jar batch_runner.jar -h
```

This generates the usage message:

```
usage: batch_runner
 -b,--batch_params <arg>    location of the batch parameters file
                            (only with -hl option)
 -c,--batch_config <arg>    location of the batch configuration file
                            (only with -hl option)
 -h,--help                  show help
 -hl,--headless             headless batch
 -model_dir <arg>           location of the model project directory
                            (needs to be specified if model
                            defined parameter types are used)
 -o,--output_dir <arg>      location of the output directory
                            (only with -hl option)
 -p,--params <arg>          location of the parameters file
                            (only with -hl option)
 -plugin_dir <arg>          location of the eclipse plugin directory
 -r,--run                   run distributed batch
                            (archive if not specified,
                            only with -hl option)
 -s,--scenario_dir <arg>    location of the scenario directory
                            (only with -hl option)
 -u,--upf <arg>             location of a custom unrolled parameter file
```

---

[4] https://emews.github.io

[5] An `unrolledParamFile.txt`, which is the result of "unrolling" a batch_parameters.xml file, is produced within a `complete_model.jar` payload during the regular payload creation process and can be used as reference.

(only with -hl and without -r option)