Lars-Erik Cederman
Laszlo Gulyas

Tutorial Sequence for RePast: An Iterated Prisoner's Dilemma Game

This document describes a tutorial sequence for beginners, developed for the software package RePast (see http://repast.sourceforge.net). Featuring five stages, the tutorial implements the core functionality of Cohen, Riolo and Axelrod (1999)'s modeling framework, which is based on an iterated prisoner's dilemma game. This modeling framework has the advantage of referring to a well-known modeling problem with obvious interest to most social scientists. Moreover, Cohen et al. explore a large number of modeling dimensions that can serve as the basis for exercises going beyond the tutorial itself. Since the tutorial builds directly on the Cohen et al. model, it is useful to consult the paper before turning to the tutorial itself.

The tutorial's stage-based logic allows for successive introduction of RePast features. The last phase includes all main ingredients of a basic RePast model, including random numbers, graphs and displays, as well as methods for batch runs. It is assumed that the user already masters basic Java. For those who want to read up on Java, Schildt (2001) and Eckel (1998) can be recommended for beginner's and more experienced programmers respectively.

The tutorial is limited to RePast's core functions. Thus, more advanced features, such as custom actions and property descriptors, are not covered. Moreover, to simplify things for the beginner, an effort has been to hide the schedule mechanism. It is our belief that schedules complicate the learning experience, although they can be useful for more advanced purposes. In order to conceal these functions, the tutorial relies on a Template class from which all the models are sub-classed rather than directly from `SimModelImpl` (see Figure 1). Customized models can be put in the place of the tutorial steps.
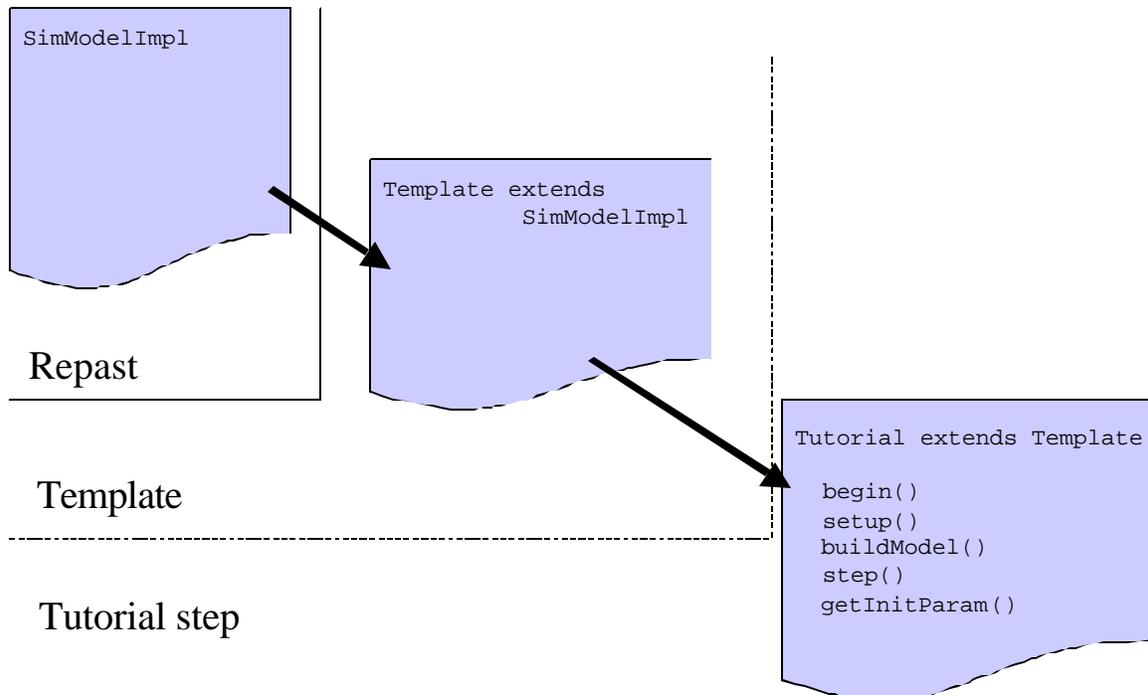
Figure 1. Using the Template to hide the schedule mechanism.

The Template sets up a schedule for the user, who is required to extend the pre-defined `buildModel()` and `step()` methods. The latter method defines what will happen in each time period by simple procedural enumeration of the execution steps rather than through the construction of a schedule hierarchy. The Template also allows the user to extend pre-defined `begin()` and `setup()` methods (cf. the RePast how-to documentation on "How to build a model"). To give the user additional control over the execution, the optional methods `getCurrentTime()` and `setStoppingTime(long v)` specify the current time step being executed and the final stopping time of batch runs respectively.

The actual tutorial sequence consists of five steps successively adding functionality to the modeling framework:
1. **SimpleIPD** sets up an iterated 2 x 2 prisoner's dilemma game and lets the user input the strategy through the RePast control panel.
2. **EvolIPD** puts the iterated game in an evolutionary setting by introducing a population of players that adapt through unstructured interaction.
3. **GraphIPD** builds on the previous step by adding a graphical user interface with a dynamically updated graph.
4. **GridIPD** abandons the "soup-like" interaction of the two previous steps in favor of a grid-based interaction topology that is shown in the graphical display.
5. **ExperIPD** extends GridIPD through the addition of batch-run capacity reading parameters from a file and outputting data to a file.

Below follows a short description of the features introduced by each tutorial step:

Step 1: SimpleIPD

The first model sets up the framework for each binary interaction between two players. When the model is run, the RePast control panel allows the user to input the strategy of each player. Assuming a total of four iterations (which is the default value), Figure 2 displays the series of payoffs received for each of the four steps together with the total sum after the iterations.

Own Strategy	Other's Strategy							
	ALLC		TFT		ATFT		ALLD	
	pay/ move	sum	pay/ move	sum	pay/ move	sum	pay/ move	sum
ALLC	3333	12	3333	12	0000	0	0000	0
TFT	3333	12	3333	12	0153	9	0111	3
ATFT	5555	20	5103	9	1313	8	1000	1
ALLD	5555	20	5111	8	1555	16	1111	4

Figure 2. The output of SimpleIPD.

There are two source files: `Model` and `Player`. Most elements in these files will be used in later stages of the tutorial. The `Model` implementation declares the model variables after which follow a series of method declarations. The most important of these are `buildModel()` and `step()`. While the former creates the two player instances and lets them know about each other, the latter pins down the exact move structure of the iterated game with a for-loop. The rest of the file contains customized definitions for RePast's control panel. The `Player` class implements the players' behavior in the `play()` method and defines the payoffs according to the `prefs[][]` matrix.

This first tutorial step introduces a number of Java features including arrays and constants that are described in Schildt (2001, Modules 5 and 7). It also illustrates the functionality control panel and the main ingredients of a RePast model (see RePast's how-to documentation under "How to use the GUI" and "How to Build a Model").

Step 2: EvolIPD

The second step of the tutorial sequence features a population rather than merely two players. Each member of the population gets to play the iterated game with a random selection of "neighbors". At the end of each time step, the players adapt by imitating the most successful strategy of the actors they met in that round.

If the initial mix of strategies is kept at the default of fourth of each, it quickly becomes obvious that this evolutionary model converges on domination by ALLD. The output in the text window lets the user monitor each strategy's progress for each time step.

Due to the strong functional similarity to SimpleIPD, this stage retains many of the code structures from the previous step. The main novelty pertains to the way that the players are handled. For this purpose, Java's collection library is used, especially the data type `ArrayList`. For more information on the collection library, see http://java.sun.com/docs/books/tutorial/collections/index.html. Another Java topic that calls for attention in this connection is casting, i.e. type conversion (see Schildt 2001, Module 2). Since the model contains a stochastic component, EvolIPD also relies on RePast's Random library (see RePast's how-to documentation: "How to work with random numbers"). Among other things, the control panel offers a way to define the seed at the beginning of each simulation. In order to make the runs more interesting to explore, though going beyond the original Cohen et al. specification, there is also a parameter `pAdapt` that slows down the rate of adaptation.

Step 3. GraphIPD
Based on EvolIPD, the third step of the tutorial sequence introduces an additional layer of implementation that takes care of the graphical user interface (GUI). This separation cleans up the model specification by separating the instrumentation from the model itself. In addition, as will be illustrated by the fifth step of the tutorial sequence, it also allows for two alternative ways of running the model: either in GUI or batch mode. For the time being, however, we content ourselves with the former.

The main feature of the GUI is a dynamically updated graph that plots the number of agents for each strategy type. Thus, this is a convenient moment to learn more about "How to create charts" (see the RePast how-to documentation). Furthermore, the Java topic of subclassing becomes particularly important at this point since the `ModelGUI` class is introduced as an extension of the `Model`. Note that the former mirrors the latter in terms of method names. For example, the GUI phase includes a `step()` method that first calls `super.step()` to make sure that the model's step is executed before the graphics procedures are invoked. A similarly nested implementation applies to `buildModel()`.[1]

---

[1] The creation of data streams for graphs in the `ModelGUI` class requires the use of somewhat exotic Java structures called interfaces, which differ from sub-classes. Whereas sub-classing extends an already existing implementation, interfaces expect the programmer to create a new implementation (see Schildt 2001, Module 8).

Step 4: GridIPD
At this point, it is time to go spatial. Instead of letting the players interact with randomly drawn players in each time period, GridIPD creates a fixed spatial grid in which the players interact with their immediate neighbors only (i.e. in the von Neumann neighborhood). This modification creates a very different dynamic that enables cooperating strategies to survive in clusters. It turns out that TFT surpasses ALLD as the most successful strategy in most runs.

The graphical addition calls for specialized visualization tools. Fortunately, RePast offers excellent display facilities (see "How to create displays" in RePast's how-to documentation). The display, which plots the players' strategies as color-coded dots, is created and updated in the `ModelGUI` along with the graph, which is retained from GraphIPD.

GridIPD also includes a probing mechanism through the addition of accessor methods at the end of the `Player` class declaration. By clicking on a player within the display, the user is able to retrieve information about that agent. Furthermore, the probing mechanism makes it possible to change the player's strategy by inserting the desired strategy's index.

Step 5: ExperIPD
The last step of the tutorial sequence equips the GridIPD model with a way to run models in batch mode in addition to the GUI features just covered. While suppressing all graphics, RePast models running in batch take their input from parameter files and produce output in separate files by invoking a data recorder (see RePast's how-to documentation under "How to use parameters and parameters files" and "How to collect data" respectively). Technically, the batch interface is created as a new sub-class `ModelBatch` extending `Model`, along the lines of `ModelGUI` in EvolIPD and GridIPD. Apart from `ModelBatch`, the rest of the model remains virtually unchanged.

How to get started
To use this tutorial, it is necessary to install RePast (see installation guide on the RePast web page). Once this is done, the tutorial and model directories need to be unzipped and put into the recommended hierarchy. For the moment, these files are provided for JBuilder users only (but can be easily changed to support the PFE editor). In the future, these zip files will hopefully become downloadable from the RePast web page. Java is very picky with file paths so it may make sense to follow the required file hierarchy as specified by Figure 3. The tree diagram, which here assumes a Windows environment, shows the main directories involved including the correct location for the Java libraries (jdk1.3) and the user's choice of editor (either pfe or the integrated developer's environment JBuilder4, see the installation instructions). We recommend that Models and Tutorial (which are highlighted in the figure) be placed inside the repast directory alongside the standard RePast sub-directories. Note that the Template will be

automatically installed as a part of the Models directory. This is also the place where users may want to put their own models.
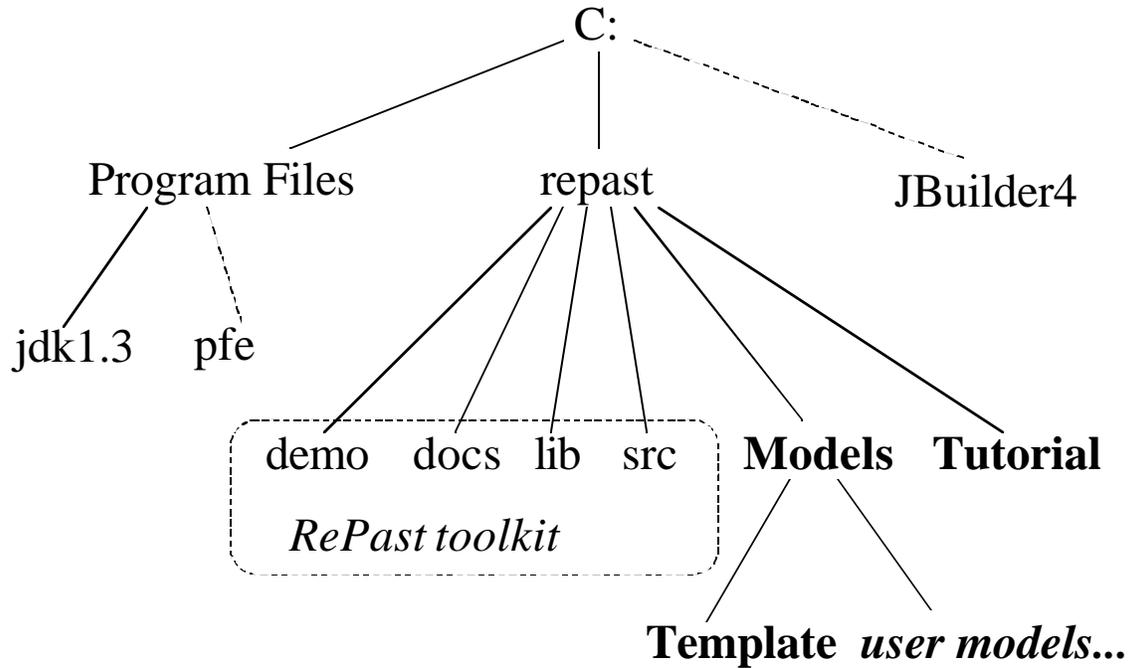

Figure 3. The recommended file structure.

References

Cohen, Michael D, Rick Riolo, and Robert Axelrod. 1999. "The emergence of social organization in the prisoner's dilemma." SFI Working Paper (see http://www.santafe/edu under publications).

Eckel, Bruce. 1998. *Thinking in Java.* Upper Saddle River: Prentice Hall.

Schildt, Herbert. 2001. *Java2: A Beginner's Guide*. Berkeley: Osborne/MacGraw Hill.