Appendix A

Implementation and Technical Issues

A.1 Random Numbers

In all instances random numbers were generated with the "ran2" generator taken from *Numerical Recipes* [138]. This generator has a period $> 2 \times 10^{18}$. It passes all known statistical tests for randomness.[1]

A.2 The SwapShop

The SwapShop program implements the artificial society described in chapter 5. It is implemented in ANSI C making use of the SunView libraries for graphical output. It has been compiled and tested on a Sun Sparc 5 (running SunOS 5.5.1) using the GNU[2] gcc compiler (version 2.0). The graphical output given in chapter 5 was produced using the

[1] In fact the authors of Numerical Recipes offer a 1000 dollar reward to anyone who can give a test that it will fail!

[2] See http://www.gnu.org/ for information concerning GNU.

SunView graphical libraries and captured using the XV[3] graphics utility program.

A.3 The StereoLab

The StereoLab program implements the artificial society described in chapter 6. It is implemented in ANSI C. It has been compiled and tested on both Sun Sparc (running SunOS 5.5.1) and PC hardware (running Linux SuSE 6.4) using the GNU gcc compiler (version 2.0). The program takes exogenous parameters (see chapter 6) as input and returns certain output measures gathered from the execution of the program (see chapter 7 and the examples in section A.5 below). The SampTool software was used to run several thousand simulation runs in various sampling modes (see chapter 8). The results were exported from SampTool into the SPSS (version 9.0 for windows) statistical package which was used to produce the graphs given in chapter 8. The program can also produce a file containing time series data for an individual simulation run. This function (along with the "-rerun" command provided by SampTool - see section A.5 below) was used to produce the individual run graphs given in chapter 8. The graphs themselves were produced within SPSS.

A.4 TagWorldII

The TagWordII program implements the artificial society described in chapter 9. It is implemented in ANSI C. It has been compiled and tested on PC hardware (running Linux SuSe 6.4) using the GNU gcc compiler (version 2.0). The program takes exogenous parameters (see chapter 9) as input and returns certain output measures gathered from the execution of the program. The SampTool (see section A.5) was used to run several simulation runs over a set of parameters. The results from these multiple runs were exported from

[3] Available as "shareware" from John Bradley, xv@devo.dccs.upenn.edu. http://www.trilon.com/xv/

SampTool into Microsoft EXCEL which was used to produce the graphs given in chapter 9 (figures 9.2 and 9.4). The program can also produce graphical output for individual runs (see chapter 9 figure 9.7). This graphical output was produced using Portable Pix Map (ppm) format libraries written by the author.

A.5 SampTool

SampTool is a general purpose software tool which allows for controlled sampling of the parameter space of an externally implemented simulation program. SampTool can make use of multiple processors distributed over a network, which share the same filestore, to execute simulation jobs in parallel. SampTool takes as input:

- A simulation specification file detailing the name and location of the external simulation program, the input parameters and output measures.
- A host file listing the names of available hosts.

When activated SampTool schedules and executes individual simulation runs on available processors and collates the results. Results are stored in a binary "Samp" file. SampTool also contains utilities to display, manipulate and convert Samp files such that individual runs my be inspected and re-run or sets of results may be exported to analysis tools such as SPSS or Microsoft EXCEL. SampTool provides the following kinds of sampling:

- Random sampling of the parameter space.
- Systematic scan of the parameter space (for both log and linear scales).
- Hill-Climbing of the parameter space based on some output measure using a specified number of independent hill-climbers for some specified number of steps.

A.5.1 Implementation

SampTool is implemented in ANSI C. It has been compiled and tested on both Sun Sparc (running SunOS 5.5.1) and PC hardware (running Linux SuSE 6.4) using the GNU gcc compiler (version 2.0). Jobs on multiple processors are executed via remote shell calls (rsh commands). Output data is collated via files written to a shared file store (each processor must share the same file store).

A.5.2 SampTool Mini-Guide

In this section a very brief guide to SampTool is given. This guide is given to indicate the general purpose nature of the software and the ease with which various kinds of sampling can be accomplished.

SampTool is invoked from the command line. Typing the following:

```
samptool -help
```

produces an on-screen summary of the commands which SampTool understands (see figure A.1). Three SampTool commands cause execution of simulations, these are the "-random", "-sweep" and "-hillclimb" commands. The other commands are utilities which facilitate the manipulation of the "samp" files, which store the results of simulation runs, produced by SampTool. In the following sections individual commands are described.

A.5.2.1 The "-random" command

The "-random" command tells SampTool to perform a set of random samples from the parameter space of a simulation (as specified in the simulation specification file).

The command has the following syntax:

```
samptool -random <SampSize> -spec <SpecNumber> -o <SampFile>
```

```
Sample Utilities - command line args are:
-----------------------------------------

-view       <sampfile> [-status <statusnum> | -notstatus <statusnum>]
-viewstats  <sampfile>
-copy       <sampfile> -o <sampfile> [-status <statusnum> | -notstatus <statusnum>]
-move       <NumToMove> <sampfile1> <sampfile2>        (move first N points from samp1 to samp2)
-merge      <sampfile> <sampfile> -o <sampfile>
-random     <SampSize> -spec <SpecNum> -o <sampfile>
-sweep      <NoOfSweeps> -spec <SpecNum> -o <sampfile>          (sweeps all points in spec)
-hillclimb  <SampSize> -steps <NoOfSteps> -outval <OutVal-to-Climb> -spec <SpecNum> -o <sampfile>
-export     <sampfile> -o <export file>                         (make spss/excel data file)
-rerun      <sampfile> <simid> -o <script file>                 (produce cmmd line to rerun sim)
-help       (gives this message)

additionally a -seed <seedvalue> can be added to –random, -hillclimb and -sweep to set seed
also -safter <int> can be added to -random and -sweep saves after every <int> finshed runs
```

Figure A.1: Output produce by the SampTool software in response to the "-help" option.

where:

<SampSize> is an integer indicating the number of random samples to make.

<SpecNumber> is an integer which identifies the specification to use from the simulation specification file.

<SampFile> is the name of the output "Samp" file in which to store the results.

example:

```
samptool -random 10000 -spec 120 -o spec120RandSimResults.samp
```

The above example tells SampTool to run the simulation identified by the number 120 in the simulation specification file 10,000 times. For each of the 10,000 simulation runs a random set of parameter values is used from the parameter space (specified in the simulation specification file). SampTool executes the simulation runs across all the hosts specified in the available hosts file. During execution, status information is displayed such as which hosts are currently running simulations, the number of completed simulations and any problems (e.g. unreachable hosts or simulation runs which have terminated incorrectly). When all the simulation runs have completed SampTool displays a message to that effect. The results of all runs are then available in the output file (in this case called

"spec120RandSimResults.samp"). The results can then be viewed or exported to other applications for analysis.

A.5.2.2 The "-sweep" command

The "-sweep" command tells SampTool to perform a complete scan or "sweep" of all unique points with the parameter space of a simulation (specified in the simulation specification file).

The command has the following syntax:

`samptool -random <NoOfSweeps> -spec <SpecNumber> -o <SampFile>`

where:

`<NoOfSweeps>` is an integer indicating the number of full sweeps to perform.

`<SpecNumber>` is an integer which identifies the specification to use from the simulation specification file.

`<SampFile>` is the name of the output "Samp" file in which to store the results.

example:

```
samptool -sweep 5 -spec 120 -o spec120SweepSimResults.samp
```

The above example tells SampTool to run the simulation identified by the number 120 in the simulation specification file 5 times for each unique point in the parameter space (specified in the simulation specification file). The total number of simulations executed therefore depends on the nature of the space specified in the specification file. SampTool executes the simulation runs across all the hosts specified in the available hosts file. During execution, status information is displayed such as which hosts are currently running simulations, the number of completed simulations and any problems (e.g. unreachable hosts or simulation runs which have terminated incorrectly). When all the simulation runs have

completed SampTool displays a message to that effect. The results of all runs are then available in the output file (in this case called "spec120SweepSimResults.samp"). The results can then be viewed or exported to other applications for analysis.

A.5.2.3 The "-hillclimb" command

The "-hillclimb" command tells SampTool to execute a number of hill-climbers over the parameter space (specified in the simulation specification file) for some number of steps.

The command has the following syntax:

`samptool -hillclimb <NoOfClimbers> -step <NoOfSteps>`

`-outval <OutValToClimb> -spec <SpecNumber> -o <SampFile>`

where:

`<NoOfHillClimbers>` is an integer indicating the number of independent hill-climbers to execute.

`<NoOfSteps>` is an integer indicating the total number of hill-climbing steps each hill-climber should execute.

`<OutValToClimb>` is an integer which identified the output value (as specified in the simulation specification file) to maximise by hill-climbing. A zero indicates the first output parameter, a one the second etc.

`<SpecNumber>` is an integer which identifies the specification to use from the simulation specification file.

`<SampFile>` is the name of the output "Samp" file in which to store the results.

example:

```
samptool -hillclimb 100 -step 200 -outval 0 -spec 120
```

`-o spec120HillclimbSimResults.samp`

The above example tells SampTool to run 100 hill-climbers for the simulation identified by the number 120 in the simulation specification file for 200 steps. This means that a total of 100 x 200 = 20,000 individual simulation runs will be executed. SampTool executes the simulation runs across all the hosts specified in the available hosts file. During execution, status information is displayed such as which hosts are currently running simulations, the number of completed simulations and any problems (e.g. unreachable hosts or simulation runs which have terminated incorrectly). When all the simulation runs have completed SampTool displays a message to that effect. The results of the *final hill-climber positions* are then available in the output file (in this case called "spec120SweepSimResults.samp"). The results can then be viewed or exported to other applications for analysis. Note: this means that only 100 samples would be saved in the output file for this example. These 100 points represent the final positions of the 100 hill-climbers.

A.5.2.4 The "-rerun" command

The "-rerun" command allows for individual runs stored in a "samp" file to be re-run. This is useful for examining individual time series of runs located, say, via hill-climbing. Each run stored within a samp file has a system generated associated unique ID number. These ID numbers can be seen when the "-view" command is executed (see figure A.2).

The command has the following syntax:

`samptool -rerun <SampFile> <SimID> -o <ScriptFile>`

where:

`<SampFile>` the name of the samp file containing the simulation run to re-run.

<SimID> is an integer which identifies the individual run within the specified samp file.

<ScriptFile> is the name of the output script file in which to store the command to re-run the selected simulation run.

example:

```
samptool -rerun spec120RandSimResults.samp 1105 -o rerunID1105
```

The above example tells SampTool to produce a script stored in the file "rerunID1105" which when executed will re-run the run stored in the samp file spec120RandSimResults.samp which has the unique ID of 1105. When the script rerunID1105 is executed the simulation will re-run with exactly the same parameter values as previously (including random number seed).

A.5.2.5 The "-export" command

The "-export" command converts a binary "samp" file produced by SampTool into a comma delimited (ASCII) text file. This format of file can be imported into most analysis applications such as SPSS, Microsoft EXCEL and Quinlan's C4.5 inductive learning system.

The command has the following syntax:

```
samptool -export <SampFile> -o <Exportfile>
```

where:

<SampFile> is the name of the samp file.

<ExportFile> is the name of the output file in which to store the comma delimited text.

example:

```
samptool -export spec120RandSimResults.samp -o spec120RandSimResults.txt
```

The above example tells SampTool to extract the simulation run details from the "spec120RandSimResults.samp" file and write them to a comma delimited text file called

"spec120RandSimResults.txt''.

A.5.2.6 The "-view" command

The "-view" command displays the contents of a samp file. Figure A.2 shows an example of the output produced by view. In addition to the individual runs stored in the samp file, the associated simulation parameters and ranges are given. The options "-status" and "-notstatus" allow for selective viewing of sim runs by status (see below).

The command has the following syntax:

`samptool -view <SampFile> [-status <StatusNo> | -notstatus <StatusNo>]`

where:

<SampFile> is the name of the samp file.

<StatusNo> is an integer specifying that only simulation runs with the given status should be displayed (after a "-status" flag) or that all runs which do not have the given status should be displayed (after a "-notstatus" flag). A status number of 1 indicates a run terminated successfully. Any other status indicates incorrect termination of the run and results from such runs should be ignored.

example:

`samptool -view spec120RandSimResults.samp -status 1`

The above command tells SampTool to display details of each simulation run stored in the file "spec120RandSimResults.samp" which terminated correctly (i.e. status = 1). SampTool produces output similar to that shown in figure A.2.

```
************** Sample details are as follows:-
======== Simulation Specification - new testing for year2000 does it work?

callmethod 1, getmethod 1, ExeCommand twt20

Inputs:

      name    type   treat     low    high     inc     fix   descs
        pt       2       1   2.000   2.000   1.000   2.000   PD outcome payoff value T
        th       2       1   0.000   0.000   1.000   0.000   PD outcome payoff value P and S
        pr       2       1   1.000   1.000   1.000   1.000   PD outcome payoff value R
        ls       1       1       8       8       1       8   number of bits for labels
        ag       1       1     100     100       1     100   number of agents
        mt       2       1   0.001   0.001   1.000   0.001   mutation rate
        gn       1       1    1000    1000       1    1000   number of generations
        rn       1       1       1       1       1       1   number of runs to average over
        su       1       1       2       2       1       2   defect start strats

Outputs:

      name    type   descs
        CC       2   average percentage of CC over whole run
        DD       2   average percentage of DD over whole run
        DC       2   average percentage of DC over whole run
        TR       2   cycle when CC becomes > 0.5 of interactions
       BCC       2   average percentage of CC over whole run
       BDD       2   average percentage of DD over whole run
       BDC       2   average percentage of DC over whole run
       ACC       2   average percentage of CC over whole run
       ADD       2   average percentage of DD over whole run
       ADC       2   average percentage of DC over whole run

============================================
All points comprising this sample are within the following region:-
---------- region ranges for region
      pt, from 2.000000 to 2.000000 in incs 1.000000 giving        1 points
      th, from 0.000100 to 0.000100 in incs 1.000000 giving        1 points
      pr, from 1.000000 to 1.000000 in incs 1.000000 giving        1 points
      ls, from        8 to        8 in incs        1 giving        1 points
      ag, from      100 to      100 in incs        1 giving        1 points
      mt, from 0.001000 to 0.001000 in incs 1.000000 giving        1 points
      gn, from     1000 to     1000 in incs        1 giving        1 points
      rn, from        1 to        1 in incs        1 giving        1 points
      su, from        2 to        2 in incs        1 giving        1 points
-------------------------------------------------------------------------
Sample contains 5 points of which 0 are bad. Points are:-
-------------------------------------------------------------
SimID 10174, on host sol14, took 10 secs, status 1, RndSeed 1567, RndGen 3 - started Wed Mar 29
19:02:43 2000
INP => pt 2.000000, th 0.000100, pr 1.000000, ls 8, ag 100, mt 0.001000, gn 1000, rn 1, su 2,
OUT => CC 0.000300, DD 0.997500, DC 0.002200, TR 0.000000, BCC 0.000000, BDD 0.000000, BDC
0.000000, ACC 0.000000, ADD 0.000000, ADC 0.000000,
-------------------------------------------------------------
SimID 10175, on host sunlab10, took 10 secs, status 1, RndSeed 5561, RndGen 3 - started Wed Mar 29
19:02:43 2000
INP => pt 2.000000, th 0.000100, pr 1.000000, ls 8, ag 100, mt 0.001000, gn 1000, rn 1, su 2,
OUT => CC 0.685600, DD 0.311700, DC 0.002700, TR 2.000000, BCC 0.105000, BDD 0.130000, BDC
0.265000, ACC 0.686100, ADD 0.311700, ADC 0.002200,
-------------------------------------------------------------
SimID 10176, on host sol13, took 10 secs, status 1, RndSeed 2501, RndGen 3 - started Wed Mar 29
19:02:43 2000
INP => pt 2.000000, th 0.000100, pr 1.000000, ls 8, ag 100, mt 0.001000, gn 1000, rn 1, su 2,
OUT => CC 0.839500, DD 0.157900, DC 0.002600, TR 2.000000, BCC 0.095000, BDD 0.195000, BDC
0.210000, ACC 0.840100, ADD 0.157600, ADC 0.002200,
-------------------------------------------------------------
SimID 10177, on host sunlab17, took 10 secs, status 1, RndSeed 8966, RndGen 3 - started Wed Mar 29
19:02:43 2000
INP => pt 2.000000, th 0.000100, pr 1.000000, ls 8, ag 100, mt 0.001000, gn 1000, rn 1, su 2,
OUT => CC 0.327200, DD 0.670100, DC 0.002700, TR 364.000000, BCC 0.000400, BDD 0.993400, BDC
0.003400, ACC 0.513400, ADD 0.484300, ADC 0.002300,
-------------------------------------------------------------
SimID 10178, on host sunlab4, took 10 secs, status 1, RndSeed 3827, RndGen 3 - started Wed Mar 29
19:02:43 2000
INP => pt 2.000000, th 0.000100, pr 1.000000, ls 8, ag 100, mt 0.001000, gn 1000, rn 1, su 2,
OUT => CC 0.866700, DD 0.131000, DC 0.002300, TR 95.000000, BCC 0.002800, BDD 0.979400, BDC
0.007300, ACC 0.956400, ADD 0.041900, ADC 0.001800,
-------------------------------------------------------------
============= stored sample stats:-
Size=10, Bad=0, Ranges: CC=[0.000200,0.997300], DD=[0.000300,0.997500], DC=[0.002200,0.002700],
TR=[0.000000,364.000000], BCC=[0.000000,0.120000], BDD=[0.000000,0.993400],
BDC=[0.000000,0.280000], ACC=[0.000000,0.998000], ADD=[0.000000,0.701100],
ADC=[0.000000,0.002300],
************** End of Sample details.
```

Figure A.2: Example output produced by SampTool with the "-view" option applied to a previously produced "samp" file.

A.5.2.7 The "-copy" command

The "-copy" command allows copies to made of a specified samp file. The optional "-status" and "-notstatus" flags allow for a copy to be made in which simulation runs which have incorrectly terminated (those with a status other than 1) are excluded. This is a useful pre-processing step before exporting using the "-export" command.

The command has the following syntax:

```
samptool -copy <SampFile1> -o <SampFile2>
[-status <StatusNo> | -notstatus <StatusNo>]
```

where:

<SampFile1> is the name of the input samp file.

<SampFile2> is the name of the output samp file.

<StatusNo> is an integer specifying that only simulation runs with the given status should be copied (after a "-status" flag) or that all runs which do not have the given status should be copied (after a "-notstatus" flag). A status number of 1 indicates a run terminated successfully. Any other status indicates incorrect termination of the run and results from such runs should be ignored.

example:

```
samptool -copy spec120RandSimResults.samp
-o spec120RandSimResultsOK.samp -status 1
```

The above command tells SampTool to copy all sim runs with a status of 1 (i.e. that terminated successfully) from the file "spec120RandSimResults.samp" to the file "spec120RandSimResultsOK.samp".

A.5.2.8 A Typical Script

Below is an example of a typical shell script combining a set of SampTools commands:

```
samptool -random 1000 -spec 52 -o spec52samp.samp

samptool -copy spec52samp.samp -o spec52sampOK.samp -status 1

samptool -export spec52sampOK.samp -o spec52sampOK.txt
```

The above script will execute 1,000 individual simulation runs with parameter values randomly selected from the parameter space specified within the simulation specifications file (specification number 52). The results are written to the file "spec52samp.samp". A copy of the output file is then made to the file "spec52sampOK.samp" excluding any simulation runs that terminated incorrectly are not copied into this new file. The file "spec52sampOK.samp" is then exported to a comma delimited text file "spec52sampOK.txt" which is suitable for loading into analysis applications.

A.5.2.9 The Simulation Specification File

SampTool assumes that a text file exists in the user's home area called "SimSpec.txt". All simulation specifications must be listed in this file. Each individual simulation specification is identified with a unique ID number (an integer) which is required by SampTool commands which require a simulation specification.

An entry in the SimSpec.txt file has the following syntax:

```
DEF <SimSpecID> <Comment> 1 1

# ''EXECMD'' <SimProgram>

# ''EXEPATH'' <SimProgramPath>

# ''RESPATH'' ./
```

INP <Pname> <Ptype> <Ptreatment> <Plow> <Phigh> <Pscale> <Pinc>

<Pfix> <Pdescription>

OUT <Pname> <Ptype> <Pdescription>

where:

<SimSpecID> is a unique integer identifier associated with the entry.

<Comment> is a double quote delimited text field containing upto 256 characters. It may contain any kind of descriptive information.

<SimProgram> is the name of an executable file which implements the simulation.

<SimProgramPath> is the path (from the user's home area) of the <SimProgram>

<Pname> is a double quote delimited text field containing upto 40 characters which identifies a parameter (a parameter name).

<Ptype> is an unquoted character field, one of either: float or int. This indicates the numerical type of the parameter.

<Ptreatment> is an unquoted character field, one of either: fix or var. This indicates if the parameter is to be treated as a fixed or variable parameter. A fixed parameter only takes on a single specified value (see <Pfix> below).

<Plow> is a numerical value specifying the lowest value that the parameter can take.

<Phigh> is a numerical value specifying the highest value that the parameter can take.

<Pscale> is an unquoted character field, one of either: lin or log. This indicates if the increments in the space are to be linear or log scale.

<Pinc> is a numerical value specifying the minimum increment allowable for the parameter (quantization increment).

<Pfix> is a numerical value specifying the fixed value a fixed parameter should take (see <Ptreatment> above).

<Pdescription> is a double quote delimited text field containing upto 256 characters. It may contain any kind of descriptive information.

example:

see figure 7.6 in chapter 7. This figure shows an example specification for the StereoLab simulation (see chapters 6.and 8).

Each IMP and OUT line is repeated for each input or output parameter. The rather awkward "#" syntax is a fix which enables backwards compatibility with previous versions of the SampTool.

A.5.2.10 The Hosts File

SampTool assumes that a text file exists in the user's home area called "hosts.txt". This text file should contain a list of the names of processors (machine names) which are to be made available to SampTool. Each host name should be on a separate line and be proceeded by the word "host". A line may contain a comment if the first character is a hash (#) symbol. A line may also be left blank. Figure A.3 shows an example hosts.txt file. Each host name should be the name of machine that can be accessed via a remote shell command (rsh).

A.5.2.11 Interfacing Simulation Programs to SampTool

When SampTool attempts to execute a simulation run it does so by executing a single command.

The command generated by SampTool has the following syntax:

<SimProgram> --ID <SimID> -RS <seed>

{-<input_param_name> <input_param_value}

```
# List of usable hosts. Note: file servers excluded.
# A hash indicates a comment line and must be first char of line

# postgrad linux

host sol1
host sol2
host sol3
host sol4
host sol5
host sol7
host sol8
host sunlab5
host sunlab6
host sunlab7
host sunlab9
host sunlab10
host sunlab11
host sunlab12
host sunlab13
```

Figure A.3: An example of a hosts.txt file. Lines beginning with a hash (#) are treated as comments and ignored. The word which follows each 'host' word is the name of a processor (machine) available via a remote shell call.

where:

<SimProgram> is the name of the simulation program as specified in the associated simulation specification (see section A.5.2.9 above).

<SimID> is a unique long integer identification number for the simulation run.

<seed> is a long integer random number seed.

<input_param_name> is the name of an input parameter as specified in the associated simulation specification (see section A.5.2.9 above).

<input_param_value> is the value for the associated <input_param_name>.

The input parameter name / value pairs are repeated for each individual input parameter (indicated above by braces).

example:

```
stereo -ID 335 -RS 433433 -S 101 -N 101 -Nc 100
```

```
-Nr 1 -T 3 -Pp 1 -Pt 5 -Pr 3 -Ps 0 -P 1 -B 8 -M 5
-Pm 0.5 -Mt 0.1 -Ci 0.5 -Cr 0.5 -Ms 0.3 -Fg 0.5
-Fc 0.6 -Fm 0.2 -Bf 0.25 -Bg 0.5 -Bc 0.2 -Tg 7
-Tc 9 -Vc 0.2 -Vg 0.3
```

The above gives an example of the kind of command SampTool might generated based on the simulation specification given in figure 7.6 in chapter 7.

In order to pass back the values of the output parameters, when the simulation has competed, the simulation program should produce an output file with the following name:

`sim<SimID>.txt`

where:

<SimID> is the same long integer value which was passed to the simulation program on the command line.

example:

```
sim335.txt
```

The above example gives the file name which would be produced for the previous example program call given above. The output file should be a text file which may contain any kind of text output but should contain *somewhere* within it a name / value word pair for each output parameter. SampTool is not fussy where these pairs are so long as the name and value are adjacent and separated by white space (spaces or tabs). Any other words in the file are ignored.

example:

```
output from simulation
CC 0.5
DD 0.25
```

```
DC 0.25
```

The above example shows the possible contents of the sim335.txt file produced by the simulation program. A value for each output parameter (as specified in the associated simulation specification - see figure 7.6 in chapter 7) is given. SampTool executes a command (as described above) then regularly checks for an output file with the appropriate name. When found, the file is read and the output values stored. The output file is then deleted.