Technical documentation of the declarative model for the South African case study

Ruth Meyer

1 Model Implementation

The declarative model is implemented in Java and Jess. It relies on the Repast library for the simulation infrastructure. The following UML class diagram shows the main classes and relationships. Repast classes are displayed in orange, Jess classes in grey.

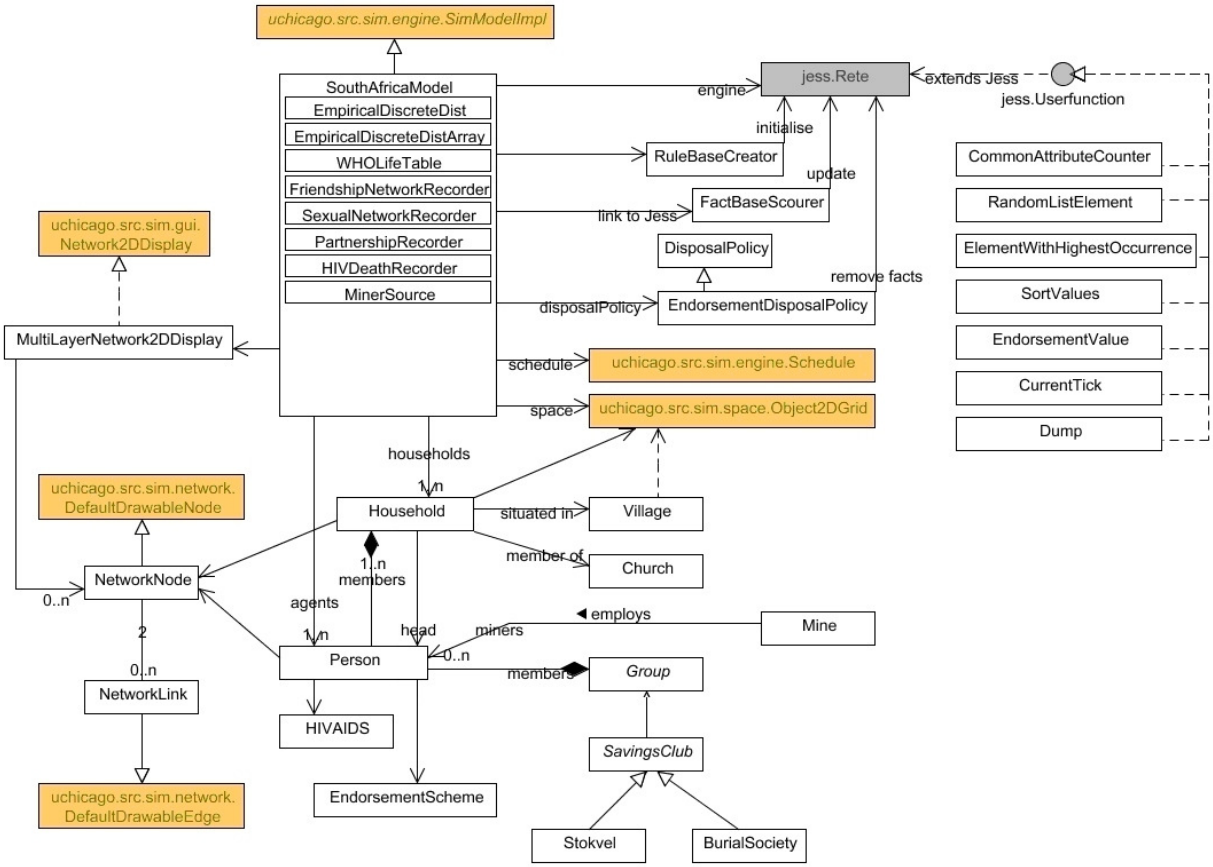

Figure 1: UML class diagram

The model logic is divided between Java and Jess. Cognition and decision processes of agents are represented (as much as possible) as rules in Jess. More procedural processes are implemented in Java. During the model development it was necessary to find a balance between the expressiveness of declarative modelling in Jess and the faster execution of Java code. Our solution has been to strive to reduce both the number of rules and the number of facts by the following measures:

1. Several rules did not deal explicitly with decision processes but were more or less procedural in nature; e.g. updating the households' cash or evolving tags. These were all ported to Java. Since some of them need to have access to the currently existing facts we implemented a Java class (*FactBaseScourer*) that browses through the fact base once per time step and delivers the necessary facts to the respective methods.

2. Whenever suitable, we replaced explicit facts with corresponding fields in Java classes acting as shadow facts in Jess. For example, facts keeping record of household economy (*monthly-food-cost, expected-income, cash-in-hand*) were replaced by fields in the *Household* class (*monthlyFoodCost, expectedIncome, cash*) and updated from Java. Rules in Jess can access these fields as slots in the shadow fact.

   This works not only for single-valued fields but for lists of values, too. The *known-person* facts keeping track of which persons a particular individual is acquainted with could thus be replaced with a *knownPerson* field in the *Person* class, which acts as a multi-slot from Jess. Instead of looking for the existence of a *known-person* fact, the respective rules now test if the person in question is a member of the *knownPersons* multi-slot. This has proven to be much faster.

3. Reducing the number of facts in Jess' memory by removing facts when they are no longer needed. For this we devised a class *DisposalPolicy*, which keeps track of which type of fact can be removed after how many ticks. To be able to be removed with a disposal policy, a fact needs to possess a slot with a time stamp, denoting its assertion time. The modeller can then specify when to remove such a fact by calling the *addPolicy()* method with the fact header, name of the time stamped slot and the wanted time lag as parameters. If no time stamped slot is specified, the default *timeStamp* is assumed. The disposal policy is executed at the end of every model step after running the Jess engine, ensuring that a time lag of 0 results in immediate removal.

   For dealing with dynamic endorsements a specialisation of Disposal Policy has been implemented. The class *EndorsementDisposalPolicy* allows for the removal of endorsements by specifying the endorsement token (e.g. *is-similar*) and the time lag. All endorsement facts possess the default *timeStamp* slot.

4. Supplying Jess with direct access to model functionality by implementing user functions instead of relying on calls on the model object. This concerns the Jess user functions *(current-tick)* and *(dump)*. The former retrieves the current model tick while the latter "dumps" the text passed as parameter on the console. The *dump* function invokes the method of the same name on the model class. Since this method only prints to the console if the respective model parameter *showOutput* is set to true, this ensures that both Jess- and Java-generated model output can be toggled via the parameter. In addition, replacing the in-built Jess output function *(printout t)* with faster output in Java achieves a speed-up.[1]

   Together, the two user functions allow for the model shadow fact to be removed from the left hand side of the majority of rules, resulting in a faster Jess execution as the complete Rete network doesn't have to be rebuilt every model tick.

In each step of the simulation, the *FactBaseScourer* is run before the Jess engine, so that

[1] With version 7 Jess provides direct access to Java's console output via *((System.out) println)*, so a mere speed-up could also have been achieved this way.

all facts are updated. After the Jess run, the *DisposalPolicy* is applied to remove all facts that fit the defined policies.

Unfortunately, the model still doesn't scale up very well. The runtime seems to increase exponentially. While it runs fast enough for 24 households, taking several hours to complete 1200 ticks, doubling the number of households (50) increases the run time to 3 days for only about 250 ticks. And that in batch mode without GUI or output to console.

2 User Guide

2.1 Model Parameters

As is common with detailed agent-based models of this kind, the declarative model has quite a few parameters. The following table shows the model's parameters and their default values.

Parameter	***Default value***	***Description***
Village/household level		
numHouseholds	24	number of initial households
numVillages	1	number of villages
numDenominations	4	number of church denomination
churchParticipation Rate	0.8	proportion of households being member of a church
adultEmploymentRate	0.4	proportion of adults with a job
government EmploymentRate	0.1	proportion of adults with a government job
standardWage	200	monthly wage within village in Rand
childGrant	200.0	monthly child grant for children of eligible age in Rand
childGrantProportion	1.0	proportion of eligible children receiving the child grant
childGrantAgeLimit	7	upper age limit for child grant
statePension	870.0	monthly state pension for seniors
pensionProportion Female	0.58	proportion of eligible female seniors receiving a pension
pensionProportion Male	0.5	proportion of eligible male seniors receiving a pension
pensionAgeLimit Female	60	lower age limit for female seniors to receive a state pension
pensionAgeLimitMale	65	lower age limit for male seniors to receive a state pension
foodCostChild	25.0	monthly food cost for a child up to 7 years in Rand
foodCostFemaleAdult	100	monthly food cost for a female in Rand
foodCostMaleAdult	120	monthly food cost for a male in Rand

Parameter	*Default value*	*Description*
schoolFee	200	yearly school fee for non-compulsory secondary school (pupils > 15 years) in Rand
collegeFee	10000	yearly university fees in Rand
minLobola	8000	minimum lobola (bride price) in Rand
maxLobola	12000	maximum lobola (bride price) in Rand
minRemittance	200	minimum remittance from a migrant agent in Rand
maxRemittance	600	maximum remittance from a migrant agent in Rand
remittanceProbability	0.1	probability for sending remittance home for any migrant in any month
burialCost	5000	average cost of a burial in Rand
shackProportion	0.4	proportion of households in the village with a shack to rent out if necessary
minShackRent	200	lower limit for the monthly rent asked for a shack in the village
maxShackRent	500	upper limit for the monthly rent asked for a shack in the village
houseCost	2000	average cost of a new house in a village in Rand
motherChildHIV TransmissionRate	0.33	mother-to-child transmission rate of HIV
birthRate	19.0	birth rate as number of births per 1000 per year
Individual level		
tagLength	7	number of tags used to model agents' characteristic traits
tagBase	5	number of different values a tag can take (integers between 0 and tagBase - 1)
maxTagEvolution Propensity	0.08	maximal tag evolution propensity
minTagEvolution Propensity	0.01	minimal tag evolution propensity; every agent is assigned a random tag evolution propensity between max and min
minEndorsementBase	1.0	minimum value for the base used to compute the overall endorsement value
maxEndorsementBase	3.0	maximum value for the base used to compute the overall endorsement value; every agent is assigned a random base between min and max
minEndorsement Classes	2	minimum number of endorsement classes
maxEndorsement Classes	5	maximum number of endorsement classes; every agent is assigned a random number of classes between min and max
upperMaxNumFriends	12	upper limit for the maximal number of friends
lowerMaxNumFriends	6	lower limit for the maximal number of friends;

Parameter	*Default value*	*Description*
		every agent is assigned a random maximal number of friends between upper and lower limit
upperMaxNum Partners	5	upper limit for the maximal number of concurrent sexual partners
lowerMaxNum Partners	1	lower limit for the maximal number of concurrent sexual partners; every agent is assigned a random maximal number of partners between upper and lower limit
similarityAgeRange	0.25	age range for similarity assessment (basically, age range for friends) given as proportion of age of the agent (e.g. age range is ± 25% of own age)
randomPartner Probability	0.1	probability to encounter a random sexual partner
Mining related		
miningTick	240	tick to start mining in the area. Set to -1 to not have any mines
neededSkilled	20	number of skilled workers needed for the mine
neededUnskilled	80	number of unskilled workers needed for the mine
mineHIVPrevalence	0.3	HIV prevalence amongst in-migrating mine workers
interArrivalTime	10	mean inter-arrival time (in ticks) for in-migrating mine workers
meanNumArrived	5	mean number of in-migrants arriving in the area at one time
sdNumArrived	2	standard deviation of the number of in-migrants
meanAgeArrived	29	mean age of in-migrants
sdAgeArrived	7	standard deviation of the age of in-migrants
Spatial environment related		
gridSizeX, gridSizeY	30	size of the 2D grid used as spatial model
neighbourhoodRadius	4.24	radius used to determine neighbourhood of a household
densityFactor	1.3	density factor, determines how "packed" villages are with houses
Simulation run related		
showGUI	false	flag to turn the GUI on (true) or off (false)
printToFile	true	flag to turn the output to file on (true) or off (false)
showOutput	true	flag to turn the output to console on (true) or off (false)
outputPath	/data/experiments/ southAfricaModel/	Path for the output directory.
seed	12345	seed for the random number generators
stopTime	2400 (50 years)	tick to stop the simulation run

2.2 Running the model from the Repast GUI

Since the model is using Repast for its simulation infrastructure, simulation runs can be started from the Repast GUI. Invoking the *main()* method of the model class (*org.cpfm.caves.za.SouthAfricaModel*) will bring up the Repast GUI (see screenshot). You can set parameters and start, pause or stop simulation runs from there.

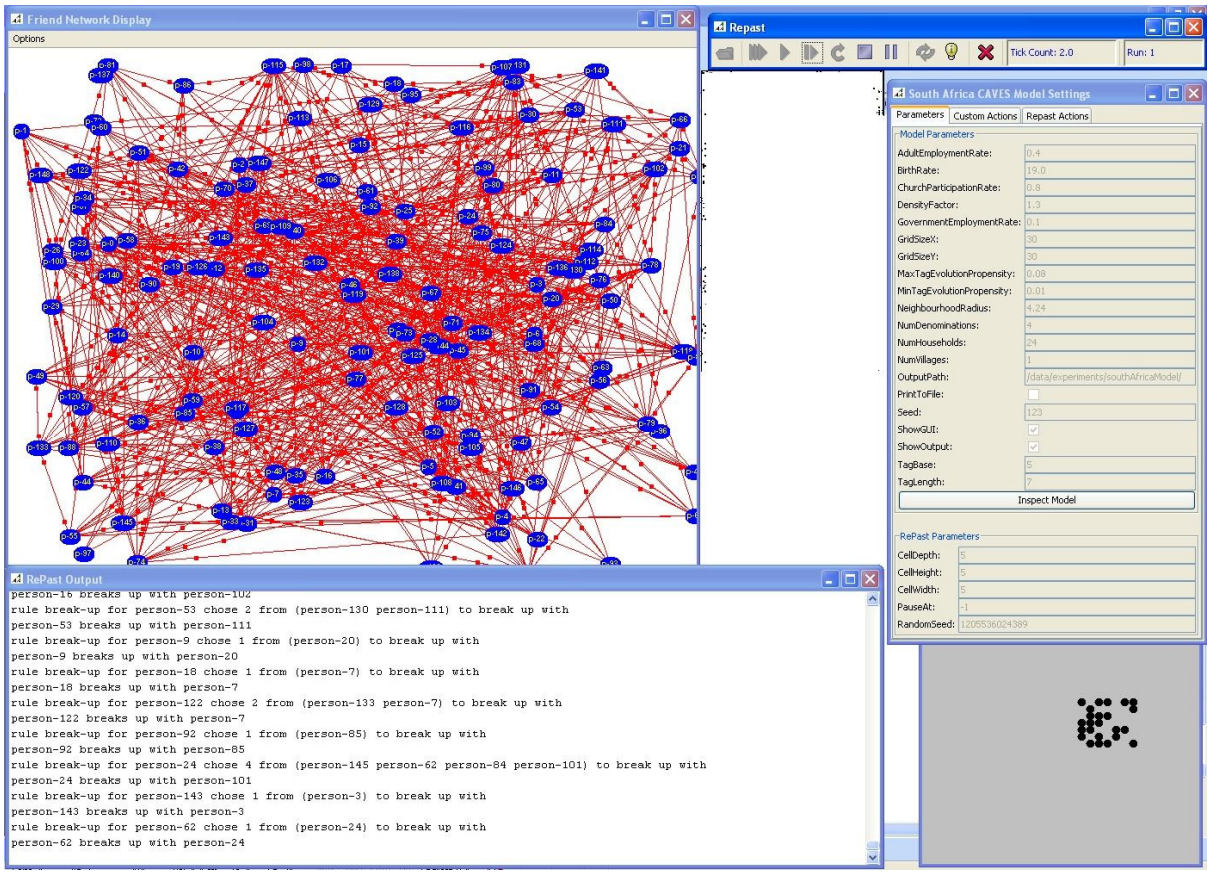

Figure 2: Screenshot of the model with GUI

It is strongly recommended to provide the Java Virtual Machine with as much memory as possible. 512 MB are necessary to run the model with 24 households.

2.3 Running the model from console as a batch run

To save on time and memory it is possible to run the model from console. This is a so-called batch run in Repast terminology and is started by invoking the Repast class *SimInit* with the model class as parameter:

```
java uchicago.src.sim.engine.SimInit -b org.cfpm.caves.za.SouthAfricaModel
```

Make sure the classpath is set correctly. The model needs the following libraries:

- flanagan.jar
- jess.jar
- repast.jar
- colt.jar

To set parameters to values different from the default values a special parameter file is needed. Please refer to the Repast documentation for details of the file format. The following shows an example parameter file for one simulation run, 50 households, a seed of 12345 and no output to console:

```
runs: 1
numHouseholds {
  set: 50
}
seed {
  set: 12345
}
showOutput {
 set_boolean: false
}
```

2.4 Model Output

The model produces output on several levels:

- *GUI*: friendship network and stokvel network (updated dynamically), kinship network, space with households in villages; it is recommended to switch this off (default).
- *Console*: model trace
- *Files*: complete model output. For each model run a folder is created in the specified output directory (default: /data/experiments/southAfricaModel/) with a unique name constructed out of the prefix "run-" and the start time of the run in system milliseconds (e.g. run-1205535622441).

  The model trace is recorded in a file called trace.txt. Time series data like number of agents, households, HIV-positive agents, households without enough food etc. on either a weekly (per tick) or monthly (every 4^{th} tick) basis is written into the files OutputWeekly.txt and OutputMonthly.txt, respectively. Data concerning deaths from HIV is recorded separately in HIVDeaths. Each year (every 48^{th} tick) the age distribution of the village population, separately for males and females, is recorded in the file AgeDistribution.txt.

  Every 10^{th} tick the friendship network is recorded in a separate network file in Pajek format. The sexual network is currently recorded for every tick in one single file. A utility to split this file into the separate Pajek network files is provided (*org.cfpm.caves.za.util.NetworkSplitter*). Data about the duration of sexual partnerships (excluding marriages) is collected in the file Partnerships.txt.

  If the model is set to include mining relevant time series data (number of in-migrated miners, number of employed unskilled workers, number of employed in-migrants, number of HIV-positive in-migrants) is recorded in OutputWeekly.txt. Each individual in-migrant is recorded in the file Mining.txt on leaving the area, noting length of stay and reason for leaving (AIDS/no job).

  In addition, several files containing data about the household economies are produced. This is a left-over from debugging.