1 BEAST - overall structure

1.1 main()

Starts at main() in class BeastMain in dr.app.beast.

Options are verbose, strict (XML), seed, overwrite (old log files), etc, plus Beagle (GPU) stuff.

Options are parsed, a console opened, dialogs done to get input file and options as required, or requested.

System.setProperty() used to store some options.

Rand seed set.

A BeastMain is created

1.2 BeastMain

Gets input file path.

Makes an infoLogger

new filereader(inputfile)

new parser(fpath, additional parsers, verbose,...)

handler for warnings and errors

logger for numerical, statistical problems

load plug-ins, and for each add its parsers.

calls parser.parse(filereader, ...) which starts things.

1.3 parser.parse()

parser is a BeastParser extending XMLParser

XMLParser is made from warnings, strict options. It can accept extra/replacement XMLObjectParsers. It can storeObjects(). It has a method parse(reader, bool run) [which was called when I looked in the debugger; there are others] which returns an ObjectStore. parse() calls an important function convert()

root = (XMLObject)convert(e, null, run, true)

root is an Element from document which is a Document built by documentBuilder.parse(in) where in is made from reader.
1.4 BeastParser

This extends XMLParser. It gets 'built-in' parsers, 'additional core' parsers, using loadProperties() to do this. It also has a setup() method. It uses addXMLObjectParser() to add these, and maybe more. So, when convert() called, all parsers are ready.

1.5 convert()

This recurively traces XML tree, matching elements to parsers and storing stuff. When it finds a Likelihood object, a Model object, or a Parameter object, it is added to a list (Likelihood.FULL_LIKELIHOOD_SET, Model.FULL_MODEL_SET, or Parameter.FULL_PARAMETER_SET). When it finds a suitable Runnable object, it starts a new thread. the object passed to the new thread is an MCMC.

1.6 MCMC

This class has three init() functions which must be called before run() [presumably one of them not all three?].

run() just calls chain().
chain() 'actually initiates MCMC analysis'.
It calls mc.runChain() where mc is a MarkovChain.

1.7 MarkovChain

MarkovChain(prior, likelihood, schedule, acceptor, ...)
schedule is the MCMC operators.
acceptor is the function for deicing whether to accept a move.
extra params are bookkeeping.

1.8 MarkovChain.RunChain()

This does
likelihood.makeDirty()
currentscore = evaluate(likelihood, prior)
currentmodel = likelihood.getModel()

while (not stopping snd not done)
    op = schedule.getNextOperator(), get mcop from op
    oldscore = currentscore
    currentModel.storeModelState()
    hastingsRatio = mcmcOperator.operate()
    if operator succeeded, score = evaluate(likelihood, prior),
    accept = acceptor(...)

    if accept
        mcop.accept(score - oldScore),
        currentmodel.acceptModelState(),
        currentscore = score
    else
        mcop.reject()
        currentmodel.restoreModelState()

... quite a bit of code to do full evaluations for tests.

Functions called by this are in Likelihood, Prior, Model, Operator

Likelihood, Prior:
likelihood.makeDirty()
evaluate(likelihood, prior) {
returns sum of prior.getLogPrior() and likelihood.getLogLikelihood()
}

Model:
storeModelState()
acceptModelState()
restoreModelState()

Operator:
mcop.accept()
mcop.reject()

1.8.1 Likelihood, Prior

Likelihood is an interface extending Loggable and Identifiable. Likelihood.Abstract provides an
abstract base class. Ignoring the loggable and identifiable bits, this looks like:

Abstract(Model model). Stores model, and calls addModelListener()
modelChangedEvent(), modelRestored(). Both just call makeDirty()
makeDirty(). Sets flag likelihoodKnown to false
getLogLikelihood(). This is final, and calls calculateLogLikelihood() iff likelihoodKnown is false.
calculateLogLikelihood(). protected, abstract, to be overridden.
Prior is an interface.
getLogPrior(). Passed a model, it returns the log prior of some aspect of it.
1.8.2 Model

Model is an interface extending Identifiable. Key methods, with comments from code:
addModelListener(). Adds a listener that is notified when the this model changes.
removeModelListener(). Removes a listener.
storeModelState(). This function should be called to store the state of the entire model. This makes the model state invalid until either an acceptModelState or restoreModelState is called.
restoreModelState(). This function should be called to restore the state of the entire model.
acceptModelState(). This function should be called to accept the state of the entire model.

1.8.3 Operator

Operator is an interface. Key methods, with comments from code:
operate(). Operates on the model, returns the hastings ratio of this operator. Throws OperatorFailedException if the operator failed and should be rejected.
accept(). Called to tell operator that operation was accepted.
reject(). Called to tell operator that operation was rejected.

SimpleMCMCOperator provides an abstract base class. There are methods for testing and tuning, ignored here. doOperation() is called by operate(), and (I think) this is the one to override. Eg TreeNodeSlide.doOperation() calls operateOneNode().

1.9 MarkovChain.RunChain() under debugger

Running *BEAST on YuleSingleLocus.xml.
The Likelihood passed to MarkovChain has two models, each with submodels.
First
- SpeciesTreeModel has SpeciesBindings
- A CompoundModel for two GammaDistributionModels
- SpeciationLikelihood

Second
- TreeLikelihood
  - TreeModel
  - GammaSiteModel, which has HKY
  - FrequencyModel
  - StrictClockBranchRates

likelihood.makeDirty() is recursive and calls MultiSpeciesCoalescent, SpeciationLikelihood, AbstractTreeLikelihood.

Then evaluate() which calls prior.getLogPrior() and likelihood.getLogLikelihood().

prior.getLogPrior() returns zero, and appears to do very little.

likelihood.getLogLikelihood() is recursive and calls MultiSpeciesCoalescent, MixedDistributionLikelihood, SpeciationLikelihood, three OneOnXPrior; then TreeLikelihood.

Then an operator is chosen, and Model.storeState() recursively calls models. Roughly like this:
AbstractModel  
SpeciesBindings does nothing  
TreeModel  
    Parameter rootHeight, something, then 31 node heights  
    copyNodeStructure  
Store ten 0.018 values, popvalues  
SpeciesTreeModel store just sets flags  
GammaDistributionModel does nothing  
variables 4.0, 0.18=popmean  
SpeciationLikelihood, store 1.0 birth rate, and likelihood  
TreeLikelihood  
    GammaSiteModel, which has HKY  
    FrequencyModel  
StrictClockBranchRates  

Then operator is called. It was a scale() operator, and was accepted. Recursive calls to model.AcceptState()

2 Multispecies coalescent code

2.1 Overview

Figure 1 shows the main ‘made of’ hierarchy.

MultiSpeciesCoalescent  
SpeciesTreeModel  
SpeciesBindings  
Parameter sppSplitPopulations  
SimpleTree spTree  
Map <,> props  
SpeciesBindings  
GeneTreeInfo[] geneTrees  
SPinfo[] species  

Figure 1: BEAST - Multispecies coalescent code

MultiSpeciesCoalescent extends Likelihood.Abstract and provides calculateLogLikelihood(). It is constructed by a parser which joins a SpeciesTreeModel and a SpeciesBindings.

SpeciesBindings ‘Binds taxa in gene trees with species information.’ SPinfo knows which taxa belong to which species. The array of GeneTreeInfo contains gene trees and popfactors.

SpeciesTreeModel is a ‘species tree which includes demographic function per branch.’ It extends AbstractModel. sppSplitPopulations is a Parameter for the populations at the ends of branches. props map ties this info to nodes in spTree.

2.2 Basic classes

- Likelihood.Abstract
- Parameter an array of doubles with extra functionality.
- AbstractModel a model made of (sub)models
**UpDownOperator** From BEAST2 doc: “This element represents an operator that scales two parameters in different directions. Each operation involves selecting a scale uniformly at random between scaleFactor and 1/scaleFactor. The up parameter is multiplied by this scale and the down parameter is divided by this scale.”

### 2.3 SpeciesBindings

This is an account of a simplified SpeciesBindings, ignoring bmPrior (Brownian motion prior) and piecewise linear stuff.

It extends AbstractModel and is constructed from an array of SPinfo (which says which taxa in which species), an array of TreeModel (gene trees), and an array of double (pop factors). The latter two are combined into an array of GeneTreeInfo.

Main fields are:

```java
GeneTreeInfo[] geneTrees;
Map<Taxon, Integer> taxon2Species = new HashMap<Taxon, Integer>();
SPinfo[] species;
boolean verbose = false;
taxon2Species is for convenience.
```

#### 2.3.1 Methods

This is most of them.

- makeCompatible
- collectCoalInfo
- nLineages
- getCoalInfo()
- swap()
- restore()
- accept()
- popFactor()
- getGeneTrees()
- handleModelChangedEvent()
- handleVariableChangedEvent()
- restoreState()
- acceptState()

### 2.4 MultiSpeciesCoalescent

Constructed from a SpeciesBindings and a SpeciesTreeModel by the very short MultiSpeciesCoalescentParser(). Constructor records the species bindings and species tree model, does addModelRestoreListener() for the species tree model, and addModelListener() for all gene trees.

It provides calculateLogLikelihood() which calls the recursive and quite complicated treeLogLikelihood(). These calculate the combined likelihood of all the gene trees given the species tree, returning $-\infty$ if a gene tree is incompatible. I think it matches this formula in Heled and Drummond (2009).
\[ \prod_{i=1}^{n} P(d_i|g_i)P(g_i|S) \]  

where \( d_i \) is data in \( i \)'th alignment, \( g_i \) is \( i \)'th gene tree, \( S \) is species tree (= topology, times, populations).

### 2.5 SpeciesTreeModel

This is an account of a simplified SpeciesTreeModel, ignoring bmPrior (Brownian motion prior) and piecewise linear stuff.

It contains a SimpleTree \( spTree \), made from a starting tree, and then adds the extra information to Nodes via a

\[
\text{Map<String, NodeProperties>} props = \text{new HashMap<String, NodeProperties>()}
\]

It is constructed (in SpeciesTreeModelParser) from a SpeciesBindings; the parameter \( \text{sppSplitPopulations} \); a starting tree; and some flags. Constructor makes calls to AbstractModel:

- \( \text{addVariable(sppSplitPopulations)} \)
- \( \text{addModel(species)} \)

It creates \( spTree \) from user tree.

### 2.6 SpeciesTreeModel methods

SpeciesTreeModel contains a lot of methods. Some important-looking ones follow.

#### 2.6.1 isCompatible

One method of this name calls a recursive one of the same name on the root node. It checks that the coalescent points are within the right branches. It compares a SpeciesBindings.CoalInfo with

\[
\text{FixedBitSet nodeSps = props.get(node).spSet}
\]

I believe the basic algorithm for one gene is

for all nodes s in species tree
  S = all the genes in the clade of s
for all nodes g in gene tree
  G = all genes in clade of g
  if ( height of g is less than height of s &&
      (G intersect S) is neither empty nor S )
    return NO
return YES

The complexity comes from making this efficient. Obviously best to find all the sets S and G first before checking pairs. \( nodeSps \) is S, and is stored in nodes of species tree. G is a CoalInfo and is stored in an array (CoalInfo[] clist) of these in a GeneTreeInfo.

#### 2.6.2 setSPsets

Given a NodeRef it returns a NodeProperties. Why ‘set’ SPsets?

#### 2.6.3 setNodeProperties

10 lines.
2.6.4 Calling hierarchy for setNodeProperties

MarkovChain.evaluate calls
likelihood.getLogLikelihood calls
treeLogLikelihood calls
getNodeDemographic calls
getProps calls
setNodeProperties (calls getDemographicPoints, setDemographics)

All this is lazy evaluation.

2.6.5 setInitialSplitPopulations

Can use ‘comments’ in Newick tree to provide these.

2.6.6 compatibleUninformedSpeciesTree

Converts / makes compatible the user-supplied start tree.

2.6.7 scale

Operator. (24 Apr 2011 not sure of this yet). This is only MCMC move on the species tree topology and node times. It has two versions. One scales all node heights and the second just one. The first is implemented here and seems simple, although I guess it is slow. The other is implemented in TreeNodeSlide. Both are based on Mau et al (1999).

2.6.8 handleModelChangedEvent

Simple, bit I guess important.

2.6.9 handleVariableChangedEvent

Simple, bit I guess important.

2.6.10 storeState

Simple, bit I guess important.

2.6.11 restoreState

More involved. Deals with changes to species tree topology and node times.

2.6.12 acceptState

Simple, bit I guess important.
2.7 TreeNodeSlide, Mau et al 1999

There is `SpeciesTreeModel.scale()` which either scales all node heights, or changes one node height using `TreeNodeSlide`. Both use idea of Mau et al 1999 [http://fisher.berkeley.edu/~rasmus/mauetal.pdf](http://fisher.berkeley.edu/~rasmus/mauetal.pdf). Looks like Mau does ± while `TreeNodeSlide` multiplies, or, if factor is negative, a uniform sample within limits. I imagine this operator was chosen because it allows topological changes but (in some sense) keeps track of nodes. Or maybe it is just a simple effective move, and the other moves are not easily available for the species tree. Called `NodeReheight` in XML.

1. Randomly orient the tree, giving each branch at each internal node a left/right label. This allows the tree to be drawn, with every node getting an (x,y) position.

2. Change one or more of the node heights.

3. Then, forgetting the identity of the internal nodes, but keeping the left-right order of the tips, construct a new tree by joining up clades in order of node height.

This will change the topology if two nodes which were originally parent and child get heights which reverse the order.

![Diagram of node reheighting](image)

Figure 2: Mau et al 1999.

This seems similar to DJ Aldous’ notion of a point process representation, or contour process, Popovic “Asymptotic genealogy...”, 2004.

2.8 MCMC ops on gene trees

There are several of these, I think any available ones can be used. This is standard BEAST calculation.

Changes to either the gene trees or the species tree may make them incompatible with one another. As far as I can see, this is handled by checking for incompatibility, and returning $-\infty$. See `isCompatible()` above.
3 Parsers

The essential elements that need to be read in are

- Data. Alignments plus meta-data (eg species bindings).
- Model, composed of sub-models.
- Likelihood made of data and model.
- Prior.
- Starting value.
- MCMC Operators.
- Logging.

MCMC chain which joins prior, likelihood, operators, logging.

This is very general, and not quite how the XML is structured. Get priors, start values inside sub-models. Border between priors and likelihoods is blurred, especially for trees.

3.1 YuleSingleLocus.XML

- taxa and alignment are straightforward and make the data. patterns follows, but they seem an implemenntation detail.
- A constantSize and a coalescentTree which make a starting gene tree.
- treeModel topology and node times for gene tree.
- strictClockBranchRates for gene tree.
- HKYModel for gene tree.
- siteModel for gene tree, which just wraps HKYModel, but could include model for site rate vaariation.
- treeLikelihood which joins patterns, treeModel, siteModel.
- species is the species bindings plus gene tree (or trees).
- sptree which contains the population parameters, a vector of split pop sizes.
- yule, the species tree prior.
- speciationLikelihood made of yule and sptree. This ‘likelihood’ appears in the prior section in mcmc.
- tmrcaStatistic For logging only.
- speciesCoalescent joins species and sptree. Appears in the prior section in mcmc.
- mixedDistributionLikelihood provides prior for pop sizes.
- operators for kappa and frequencies in HKY, scaling whole thing, gene tree moves, species tree moves.
- mcmc with priors speciesCoalescent, mixedDistributionLikelihood, speciationLikelihood, and for kappa, popMean, yule birth rate. Likelihood is treeLikelihood for gene tree. Lots of logging.
- report - a timer.
3.2 List of parsers particular to *BEAST

3.2.1 SpeciesBindingsSPinfoParser

Reads taxa in a species, used by next.

3.2.2 SpeciesBindingsParser

Reads SpeciesBindingsSPinfos, and GeneTreeInfos made of trees and pop factors, latter not in YuleSingleLocus.

```xml
<species id="species">
  <sp id="C_olivacea">
    <taxon idref="AF_110423_C_olivacea"/>
    <taxon idref="AF_109015_C_olivacea"/>
  </sp>
  ...
  <geneTrees id="geneTrees">
    <treeModel idref="treeModel"/>
  </geneTrees>
</species>
```

3.2.3 SpeciesTreeModelParser

Reads boolean constantRoot, parameter speciesTree.splitPopSize and initial value (0.18).

3.2.4 MultiSpeciesCoalescentParser

Short. Joins a SpeciesBindings and a SpeciesTreeModel.

3.2.5 SpeciationLikelihoodParser

Joins Yule prior and SpeciesTree.

```xml
<speciationLikelihood id="speciation.likelihood">
  <model>
    <yuleModel idref="yule"/>
  </model>
  <speciesTree idref="sptree"/>
</speciationLikelihood>
```

3.2.6 TreeNodeSlideParser

Operator. nodeReHeight for species tree moves.

3.2.7 SpeciesTreeSimplePriorParser

Not used in YuleSingleLocus.XML.
3.2.8 SpeciesTreeStatisticParser

'A statistic that returns true if the given population tree is compatible with the species tree. Compatibility is defined as the compatibility of the timings of the events, so that incompatibility arises if two individuals in the population tree coalesce before their species do in the species tree.'

Not used in YuleSingleLocus.XML.