Contents

Executive summary **4**

1 Use Case - Data bases (DB) **5**

1.1 Attributes 6

1.2 DataInstance 7

1.3 Data file reader 8

1.4 Scalable data base management 9

2 Use Case - Basic data structures (BS) **10**

2.1 Static variables 11

2.2 Dynamic variables 12

2.3 Directed acyclic graph (DAG) 13

2.4 Distributions 14

2.5 Bayesian network 16

2.6 Dynamic DAG 17

2.7 2T-DBN 18

3 Use Case - Hugin link (HL) **19**

3.1 Converter from Amidst to Hugin 20

3.2 Converter from Hugin to Amidst 21

4 Use Case - Importance sampling (IS) **22**

5 Use Case - Maximum likelihood (ML) **23**

5.1 Exponential family distributions 24

6 Use Case - Variational message passing (VMP) **25**

7 Use Case - Expectation propagation (EP) **26**

8 Use Case - MAP with deterministic approximations (DMAP) **27**

9 Use Case - Variational MAP inference (VMAP) **28**

10 Use Case - TAN classifier (TAN) **29**

11 Use Case - Parallel PC (PPC) **30**

12 Use Case - Dynamic classifiers (DC) **31**

13 Use Case - Feature selection (FS) **32**

Class diagrams **33**

13.1 Package eu.amidst.core 33
13.2 Package eu.amidst.core.database 34
13.3 Package eu.amidst.core.database.dynamics 35
13.4 Package eu.amidst.core.database.readers 35
13.5 Package eu.amidst.core.database.filereaders 36
13.6 Package eu.amidst.core.database.arffFileReader 37
13.7 Package eu.amidst.core.database.arffWekaReader 37
13.8 Package eu.amidst.core.variables 38
13.9 Package eu.amidst.core.distribution 39
13.10Package eu.amidst.core.modelstructure 40
13.11Package eu.amidst.core.huginlink 41
13.12Package eu.amidst.core.potential 42
13.13Package eu.amidst.core.exponentialfamily 43
13.14Package eu.amidst.core.utils 44

Document history

Version	Date	Author (Unit)	Description
v0.3			First draft finished

Executive summary

The aim of this document is to track the development process of the AMIDST toolbox.

The document is structured into sections, such that each section includes the required details for a specific use case, i.e., a general description of the purpose of the use case as well as a categorised list of its associated requirements.

We then associate to each use case a list of so-called ***functionalities*** (included in independent subsections) that are coded in the toolbox to cover the specific use case and their associated requirements. A *functionality*, or also called a *feature*, is a set of java classes that allows to perform a specific task or that defines a coherent concept within the toolbox. For example, the creation and managment of random variables, reading a data set from a file, building and handling dynamic Bayesian networks, the maximum likelihood estimation, etc. The set of *functionalities* identify those key parts of the toolbox that any developer or toolbox user need to understand in order to make a proper use of the AMIDST toolbox.

Another important part of this document is to detail the time line of the toolbox development. Each of the use cases contain an additional subsection named *Time tables* which contain details about the development phase of each *functionality* associated to this use case. Six main phases are identified: design, prototype, code-review, testing, documentation and (first) release. Through this section, we can easily track the evolution of the use case.

In the last section, we detail the class diagram of each package in the AMIDST toolbox.

We highlight that the contents detailed here can be later modified or re-organized to create the different deliverables and the final user-manual of the AMIDST toolbox.

1 Use Case - Data bases (DB)

Priority: Must

Deadline: M15

Responsible: Sigve

Description of the use case

This use case covers the design description and management of data bases that will be used by Amidst learning and inference algorithms implemented in the toolbox.

Must-Requirements list of the use case

1. Data on memory
2. Data on disk

Should-Requirements list of the use case

1. Data on stream

Could-Requirements list of the use case

1. Parallel data base

In the following sections, the main features pertaining to this use case, namely, DataInstance, Scalable data base management, DataFileReader, and attributes are presented. For each functionality, we include a short description, the corresponding detailed functionality, as well as a code example containing a brief example using the described feature. Note that more code examples could be found in the package `eu.amidst.examples`.

1.1 Attributes

Deadline: M12

Responsible: Sigve

Package: `eu.amidst.core.databases`

Description

Attributes serve as an intermediary to build either static or dynamic variables which are parsed from the input dataset and/or directly specified by the user (in particular using an additional class that extends this class).

Detailed functionality

- List of objects of the type Attribute. This list becomes Unmodifiable after construction.
- There can be two special attributes, namely, "TIME_ID" AND "SEQUENCE_ID". The former refers to a temporal identifier, whereas the second identifies a particular sequence (e.g., a client in CajaMar or a drill in Verdande). They are only used whenever they explicitly appear in the dataset with that particular names.

Code example

```
DataOnDisk data = new DynamicDataOnDiskFromFile(new WekaDataFileReader(
new String(‘‘datasets/syntheticDataVerdandeScenario1.arff’’)));

Attribute attTRQ = data.getAttributes().getAttributeByName(‘‘TRQ’’);
Attribute attROP = data.getAttributes().getAttributeByName(‘‘ROP’’);

List<Attribute> attributeList = new ArrayList();
attributeList.add(attTRQ);
attributeList.add(attROP);
```

1.2 DataInstance

Deadline: M12

Responsible: Sigve

Package: `eu.amidst.core.database`

Description

DataInstance refers to a row in the data set typically containing a value assignment for each attribute. In terms of code, DataInstance is defined as an interface that is implemented by either a StaticDataInstance or a DynamicDataInstance. The former stores one row (DataRow) of the data set at a time, whereas the second stores two rows of the data set (one for the past and another for the present).

Detailed functionality

A DynamicDataInstance always has a TimeID and a SequenceID. If this two attributes, or any of the two, are not in the dynamic data set, then they are automatically filled in, incrementally for the TimeID and with a value of 1 for the SequenceID.

- StaticDataInstance: only contains a single DataRow.
- DynamicDataInstance: contains a past and a present DataRow, a TimeID, and a SequenceID. The TimeID attribute in the data set can be used to represent missing samples (stored as DataRowMissing), in which all attribute values are NaN.

Code example

1.3 Data file reader

Deadline: M12

Responsible: Sigve

Package: `eu.amidst.core.database.filereaders`

Description

It refers to the different parsers to be used to convert Arff (Attribute-Relation File Format) data files (or potentially other data file types such as csv) into data instances and attributes.

Detailed functionality

Currently, we have the following two readers:

1. ArffWekaReader: fully working and tested using Weka parser.
2. ArffFileReader: not finished yet, but it is expected to include the AMIDST parser.

Code example

```
DataOnDisk data = new DynamicDataOnDiskFromFile(new WekaDataFileReader(
new String(‘‘datasets/syntheticDataVerdandeScenario1.arff’’)));
```

1.4 Scalable data base management

Deadline: M12

Responsible: Sigve

Package: `eu.amidst.core.database`

Description

This functionality basically define how to manage the output of the data file reader (i.e., the DataRows) and convert them into DataInstances. It is crucial to distinguish here between whether we are dealing with the static or dynamic case, as well as whether the data could be loaded into memory, read from disk, or processed as a stream.

Detailed functionality

- Static:
  - Data on memory
  - Data on disk
  - Data on stream
- Dynamic:
  - Data on memory
  - Data on disk
  - Data on stream

Code example

2 Use Case - Basic data structures (BS)

Priority: Must

Deadline: M15

Responsible:

Description of the use case

This use case will cover the basic data structures to handle the probabilistic graphical models belonging to the AMIDST model class, and to be included in the toolbox (a tentative list is given below).

On the other hand, information about possible models to be plug-in into the toolbox beyond the project is also desirable to be indicated.

Must-Requirements list of the use case

1. Static Bayesian network (BN)
2. Two-time slice dynamic Bayesian network (2T-DBN)
3. Bounded dynamic Bayesian network

Should-Requirements list of the use case

1. Additional operations for learning Bayesian networks

Could-Requirements list of the use case

1. Factor graphs

2.1 Static variables

Deadline: M12

Responsible: Andres

Code-Package: `eu.amidst.core.variables`

Description

Static variables define the list of static variables to be used in the static BN models.

Detailed functionality

- List of objects of type `Variable`.
- A static variable is characterised by its name, ID, the number of states, the state space type, the distribution type, as well as if it is observable or not.
- The state space type could be either Multinomial or Real.
- The distribution type could be either Multinomial or Gaussian.
- The list of observable static variables is initialised using the list of Attributes (that are already parsed from the dataset or specified by the user), then hidden variables can be also added.

Code example

```
StaticVariables variables = new StaticVariables(data.getAttributes());
Variable A = variables.getVariableByName("A");
Variable B = variables.getVariableByName("B");
Variable C = variables.getVariableByName("C");

VariableBuilder variableBuilder = new VariableBuilder();
variableBuilder.setName("HiddenVar");
variableBuilder.setObservable(false);
variableBuilder.setStateSpace(new
      MultinomialStateSpace(Arrays.asList("TRUE","FALSE")));
variableBuilder.setDistributionType(DistType.MULTINOMIAL);
Variable hidden = variables.addHiddenVariable(variableBuilder);
```

2.2 Dynamic variables

Deadline: M12

Responsible:

Code-Package: `eu.amidst.core.variables`

Description

Dynamic variables define the list of dynamic variables to be used in the dynamic BN models.

Detailed functionality

- List of objects named allVariables and temporalClones of the type `Variable`.
- A dynamic variable is characterised by its name, ID, the number of states, the state space type, the distribution type, if it is observable or not, and if it is temporal clone or not.
- The state space type could be either Multinomial or Real.
- The distribution type could be either Multinomial or Gaussian.
- The list of observable dynamic variables and their temporal clones is initialised using the list of Attributes (that are already parsed from the dataset or specified by the user), then hidden variables and their temporal clones can be also added.

Code example

```
DynamicVariables dynamicVariables = new DynamicVariables();
Variable observedROP = dynamicVariables.addObservedDynamicVariable(attROP);
Variable observedTRQ = dynamicVariables.addObservedDynamicVariable(attTRQ);
Variable realTRQ = dynamicVariables.addRealDynamicVariable(observedTRQ);
VariableBuilder variableBuilder = new VariableBuilder();
variableBuilder.setName("HiddenVar");
variableBuilder.setObservable(false);
variableBuilder.setStateSpace(new RealStateSpace());
variableBuilder.setDistributionType(DistType.GAUSSIAN);
Variable hidden = dynamicVariables.addHiddenDynamicVariable(variableBuilder);
```

2.3 Directed acyclic graph (DAG)

Deadline: M12

Responsible: Hanen

Code-Package: `eu.amidst.core.models`

Description

The class Directed acyclic graph (DAG) defines the Bayesian network graphical structure over a list of static variables.

Detailed functionality

- It defines the parent set for each variable.
- It test and detect if a DAG contains cycles or not.

Code example

```
WekaDataFileReader reader = new
      WekaDataFileReader("data/dataWeka/contact-lenses.arff");
StaticVariables variables = new StaticVariables(reader.getAttributes());
DAG dag = new DAG(variables);

StaticVariables variables = dag.getStaticVariables();
Variable A = variables.getVariableById(0);
Variable B = variables.getVariableById(1);
Variable C = variables.getVariableById(2);
Variable D = variables.getVariableById(3);
Variable E = variables.getVariableById(4);

dag.getParentSet(B).addParent(A);
dag.getParentSet(C).addParent(A);
dag.getParentSet(C).addParent(B);
dag.getParentSet(D).addParent(B);
dag.getParentSet(E).addParent(B);
```

2.4 Distributions

Deadline: M15

Responsible: Antonio Fernández

Code-Package: `eu.amidst.core.distributions`

Description

This functionality addresses the set of conditional probability distributions considered to be included in the toolbox. Variables with Gaussian and multinomial distributions are modeled. The variables arrangement in the model structure gives rise to the different types of probability distributions, one for each variable in the network.

This functionality is tightly connected to functionality `Variable` and `DAG` to know both the type and the set of parents of each variable.

Detailed functionality

The type and the set of parents of each variable determine the different defined probability distributions as follows:

- Multinomial variable with no parents
- Multinomial variable with multinomial parents.
- Gaussian variable with no parents.
- Gaussian variable with multinomial parents.
- Gaussian variable with Gaussian parents.
- Gaussian variable with a mixture of multinomial and Gaussian parents.

Note that a multinomial variable is not allowed to have Gaussian parents and therefore it has not been included in the list above.

Multinomial parents are only used for indexing the set of possible distributions of the variable, so the functionality when no multinomial parents reduces to the general case.

Code example

This is brief code fragment showing the definition of the distribution for a variable `var` given the set of its parents:

```
ParentSet parentSet = this.getDAG().getParentSet(var);
int varID = var.getVarID();
this.distributions[varID]=
     DistributionBuilder.newDistribution(var, parentSet.getParents());
parentSet.blockParents();
```

2.5 Bayesian network

Deadline: M12

Responsible: Hanen

Code-Package: `eu.amidst.core.models`

Description

This class defines a static Bayesian network using the already specified graphical structure (DAG) along with the conditional probability distribution of each variable given the set of its parents.

Detailed functionality

- The distribution of each variable in the Bayesian network is initialised and specified according to its type and the type of its potential parent set.
- After this step, the set of parents of each variable becomes unmodifiable.

Code example

This is brief code fragment showing the definition of a Bayesian network using the previously created `dag`. It automatically looks at the distribution type of each variable and their parents to initialise the Distributions objects that are stored inside (i.e., Multinomial, Normal, CLG, etc). The parameters defining these distributions are correspondingly initialised.

```
BayesianNetwork bn = BayesianNetwork.newBayesianNetwork(dag);
```

2.6 Dynamic DAG

Deadline: M12

Responsible: Hanen

Code-Package: `eu.amidst.core.models`

Description

It defines the dynamic Bayesian network graphical structure over a list of dynamic variables by specifying their parent sets at time 0 and time T.

Detailed functionality

- It defines the parent set at time 0 and time T for each dynamic variable.
- It test and detect if a dynamic DAG contains cycles or not.

Code example

```
DynamicDAG dynamicDAG = new DynamicDAG(dynamicVariables);

dynamicDAG.getParentSetTimeT(observedTRQ).addParent(observedWOB);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(observedRPMB);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(observedMFI);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(realTRQ);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(hidden);
dynamicDAG.getParentSetTimeT(observedTRQ).addParent(mixture);
```

2.7 2T-DBN

Deadline: M12

Responsible:

Code-Package: `core.models`

Description

This functionality is similar to the one defined for Bayesian network (see 2.5), but it aims here to represent instead a two slice-time dynamic Bayesian network (2T-DBN). It handles its structure (dynamic DAG defined over a set of dynamic variables) and the set of its distributions at time 0 and time T.

Detailed functionality

- The distributions of each dynamic variable at both time 0 and time T are initialised and specified according to the variable type and the type of its potential parent set.
- After this step, the set of parents of each dynamic variable becomes unmodifiable.

Code example

This is brief code fragment showing the definition of a dynamic Bayesian network using the previously created `dynamicDAG`. It automatically looks at the distribution type of each variable and their parents to initialise the Distributions objects that are stored inside (i.e., Multinomial, Normal, CLG, etc). The parameters defining these distributions are correspondingly initialised.

```
DynamicBayesianNetwork dynamicBayesianNetwork =
 DynamicBayesianNetwork.newDynamicBayesianNetwork(dynamicDAG);
```

3 Use Case - Hugin link (HL)

Priority: Must

Deadline: M15

Responsible: A. Fernández

Description of the use case

This use case contains all the functionality needed to link the AMIDST toolbox with the HUGIN software. This linkage is addressed by converting Hugin models into AMIDST models, and vice versa. This feature is extremely useful as it allows expanding the testing possibilities of the AMIDST models within a well-stablished platform as Hugin. Also, the link to Hugin can be used for providing some extra functionality to AMIDST that will not be implemented. Finally, the linkage is useful for comparison purposes, i.e., a new inference algorithm implemented in AMIDST could be compared with some state-of-the-art algorithm included in Hugin.

Must-Requirements list of the use case

1. Bayesian network converter from AMIDST to Hugin format.
2. Bayesian network converter from Hugin to AMIDST format.
3. Possibility of saving the converted Hugin network in a `.net` file.

Could-Requirements list of the use case

1. Converter from AMIDST to HUGIN and vice versa of some functionality that is not relevant for model representation.

3.1 Converter from Amidst to Hugin

Deadline: M12

Responsible: A. Fernández

Code-Package: `core.huginlink`

Description

This functionality addresses the conversion of a Bayesian network from Amidst to Hugin. This conversion is done at "object-level", which is far more efficient that if done by converting the models to data files and, then, parsing them.

Detailed functionality

Code example

This is brief code fragment showing how to convert a BN from Amidst to Hugin format and stored it on a file. Then, we can open HUGIN and visually inspect the BN created with the AMIDST toolbox.

```
ConverterToHugin converterToHugin = new ConverterToHugin(bn);
converterToHugin.convertToHuginBN();
converterToHugin.getHuginNetwork().saveAsNet
                 ("networks/huginStaticBNHiddenExample.net");
```

3.2 Converter from Hugin to Amidst

Deadline: M12

Responsible: A. Fernández

Code-Package: `core.huginlink`

Description

This functionality addresses the conversion of a Bayesian network from Hugin to Amidst. This conversion is done at "object-level", which is far more efficient that if done by converting the models to data files and, then, parsing them.

Detailed functionality

Code example

This is brief code fragment showing how to convert a BN from Hugin to Amidst.

```
ParseListener parseListener = new DefaultClassParseListener();
this.huginBN = new Domain ("networks/huginNetworkFromAMIDST.net",
                           parseListener);
ConverterToAMIDST converter = new ConverterToAMIDST(this.huginBN);
converter.convertToAmidstBN();
this.amidstBN = converter.getAmidstNetwork();
```

4 Use Case - Importance sampling (IS)

Priority: Must

Deadline: M16

Responsible:

Description of the use case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements list of the use case

1. Include short description

Should-Requirements list of the use case

1. Include short description

Could-Requirements list of the use case

1. Include short description

5 Use Case - Maximum likelihood (ML)

Priority: Must

Deadline: M12

Responsible:

Description of the use case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements list of the use case

1. Include short description

Should-Requirements list of the use case

1. Include short description

Could-Requirements list of the use case

1. Include short description

5.1 Exponential family distributions

Deadline: M12

Responsible: Andres

Code-Package: `core.exponentialfamily`

Description

This functionality addresses the representation of the probability distributions in an exponential family form. I.e. they are encoded by a vector of natural and moment parameters. Functionality for computing the sufficient statistics for a data instance is also provided. These set of classes are going to be used across several use cases: ML, VMP, EP, VMAP, etc.

Detailed functionality

The type of each variable and its parents determine the different exponential family distributions detailed next:

- Multinomial variable with no parents
- Multinomial variable with multinomial parents.
- Gaussian variable with no parents.
- Gaussian variable with multinomial parents.
- Gaussian variable with Gaussian parents.
- Gaussian variable with multinomial and Gaussian parents.

Code example

6 Use Case - Variational message passing (VMP)

Priority: Must

Deadline: M15

Responsible:

Description of the Use Case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements List of the Use Case

1. Include short description

Should-Requirements List of the Use Case

1. Include short description

Could-Requirements List of the Use Case

1. Include short description

7 Use Case - Expectation propagation (EP)

Priority: Must

Deadline: M18

Responsible:

Description of the Use Case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements List of the Use Case

1. Include short description

Should-Requirements List of the Use Case

1. Include short description

Could-Requirements List of the Use Case

1. Include short description

8 Use Case - MAP with deterministic approximations (DMAP)

Priority: Must

Deadline: M17

Responsible:

Description of the Use Case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements List of the Use Case

1. Include short description

Should-Requirements List of the Use Case

1. Include short description

Could-Requirements List of the Use Case

1. Include short description

9 Use Case - Variational MAP inference (VMAP)

Priority: Should

Deadline: M28

Responsible:

Description of the Use Case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements List of the Use Case

1. Include short description

Should-Requirements List of the Use Case

1. Include short description

Could-Requirements List of the Use Case

1. Include short description

10 Use Case - TAN classifier (TAN)

Priority: Must

Deadline: M18

Responsible:

Description of the Use Case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements List of the Use Case

1. Short Description

Should-Requirements List of the Use Case

1. Short Description

Could-Requirements List of the Use Case

1. Short Description

11 Use Case - Parallel PC (PPC)

Priority: Must

Deadline: M17

Responsible:

Description of the Use Case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements List of the Use Case

1. Short Description

Should-Requirements List of the Use Case

1. Short Description

Could-Requirements List of the Use Case

1. Short Description

12 Use Case - Dynamic classifiers (DC)

Priority: Could

Deadline: M29

Responsible:

Description of the Use Case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements List of the Use Case

1. Short Description

Should-Requirements List of the Use Case

1. Short Description

Could-Requirements List of the Use Case

1. Short Description

13 Use Case - Feature selection (FS)

Priority: Must

Deadline: M20

Responsible:

Description of the Use Case

Enter a textual description of the use case. Link to other use cases or functionalities if needed.

Must-Requirements List of the Use Case

1. Include short description

Should-Requirements List of the Use Case

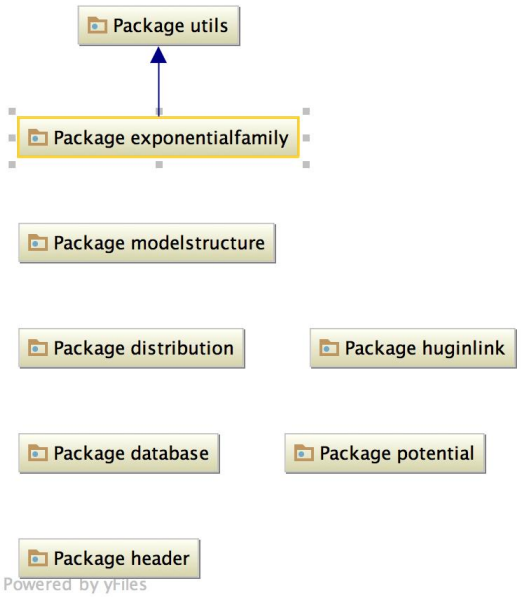
1. Include short description

Could-Requirements List of the Use Case

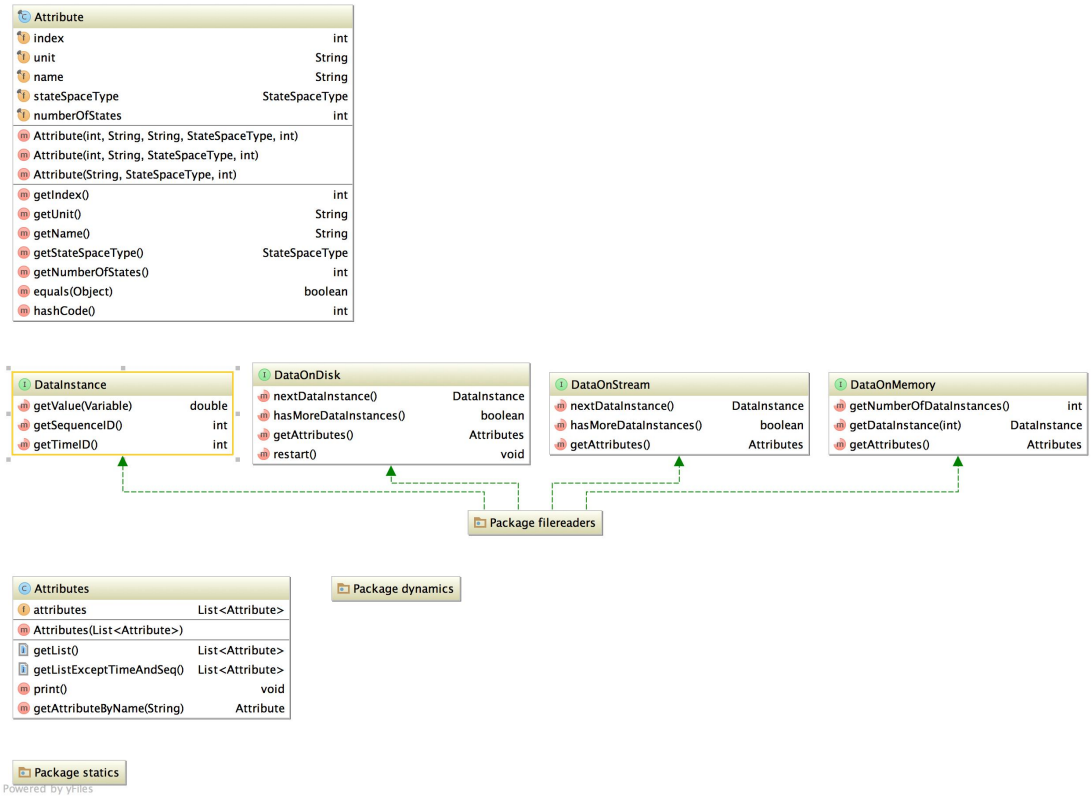
1. Include short description

Class diagrams

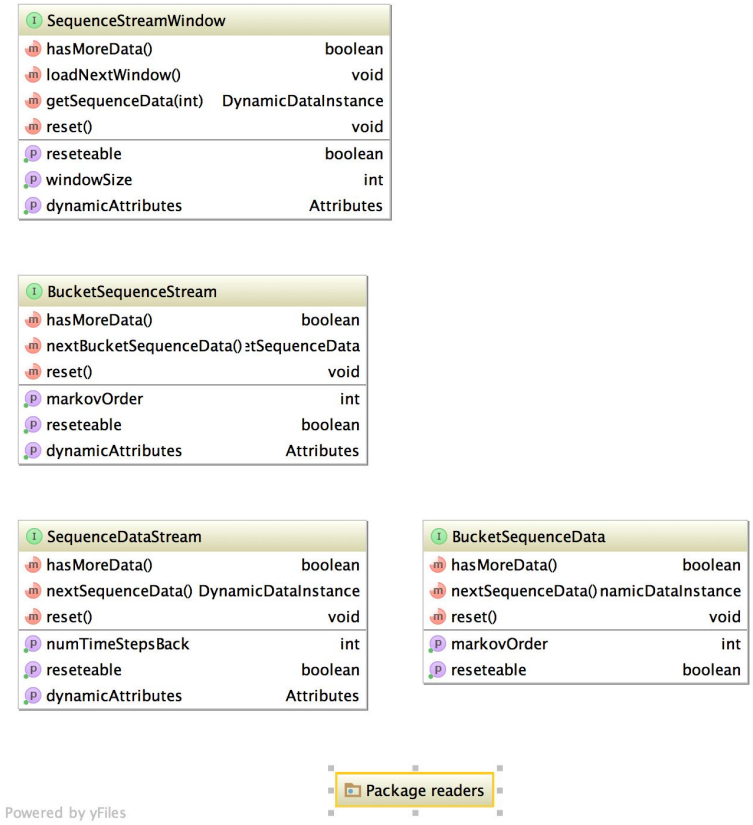
13.1 Package eu.amidst.core



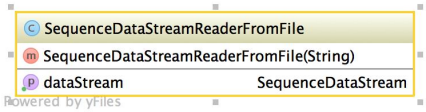
13.2 Package eu.amidst.core.database



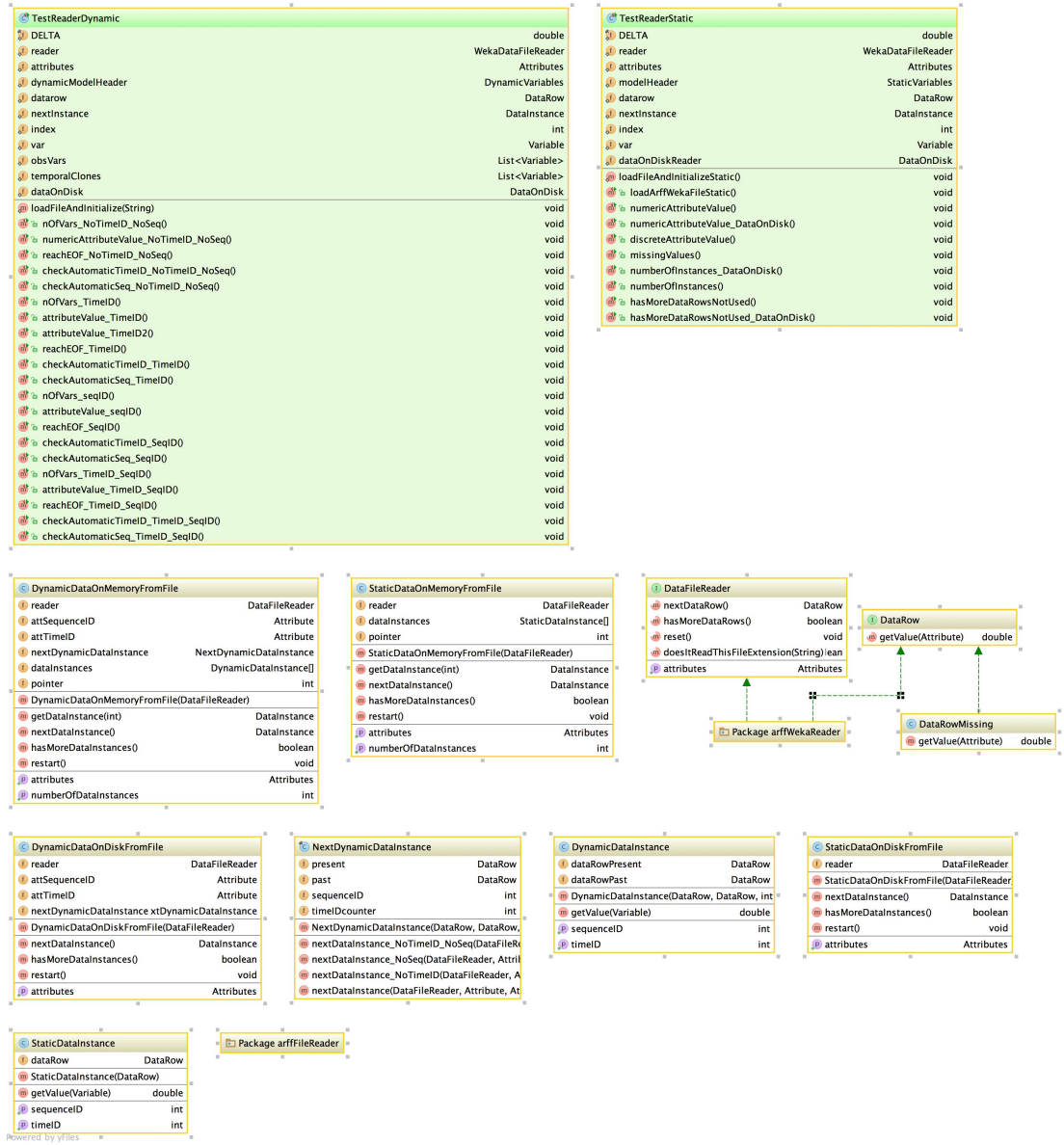
13.3 Package eu.amidst.core.database.dynamics



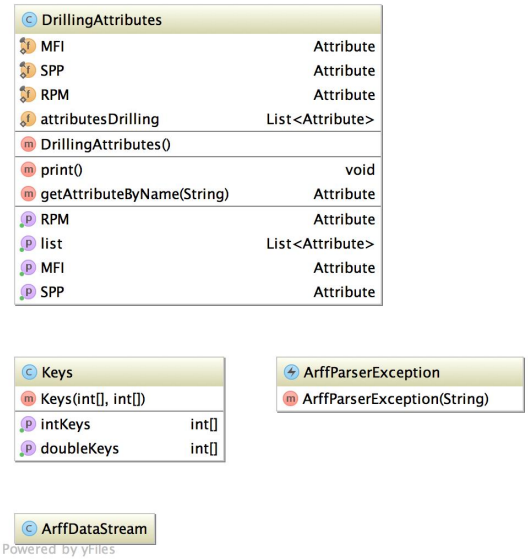
13.4 Package eu.amidst.core.database.readers



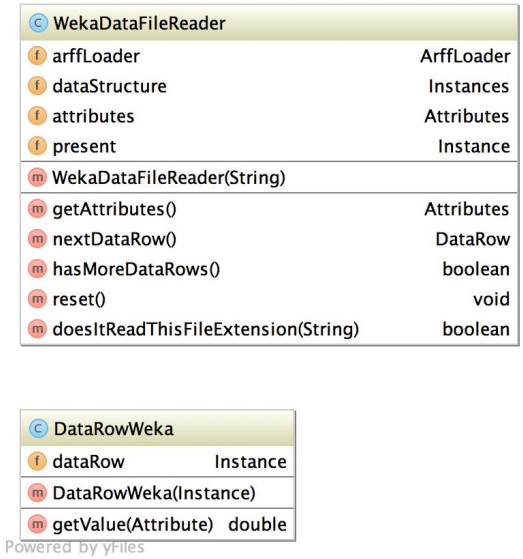
13.5 Package eu.amidst.core.database.filereaders



13.6 Package eu.amidst.core.database.arffFileReader

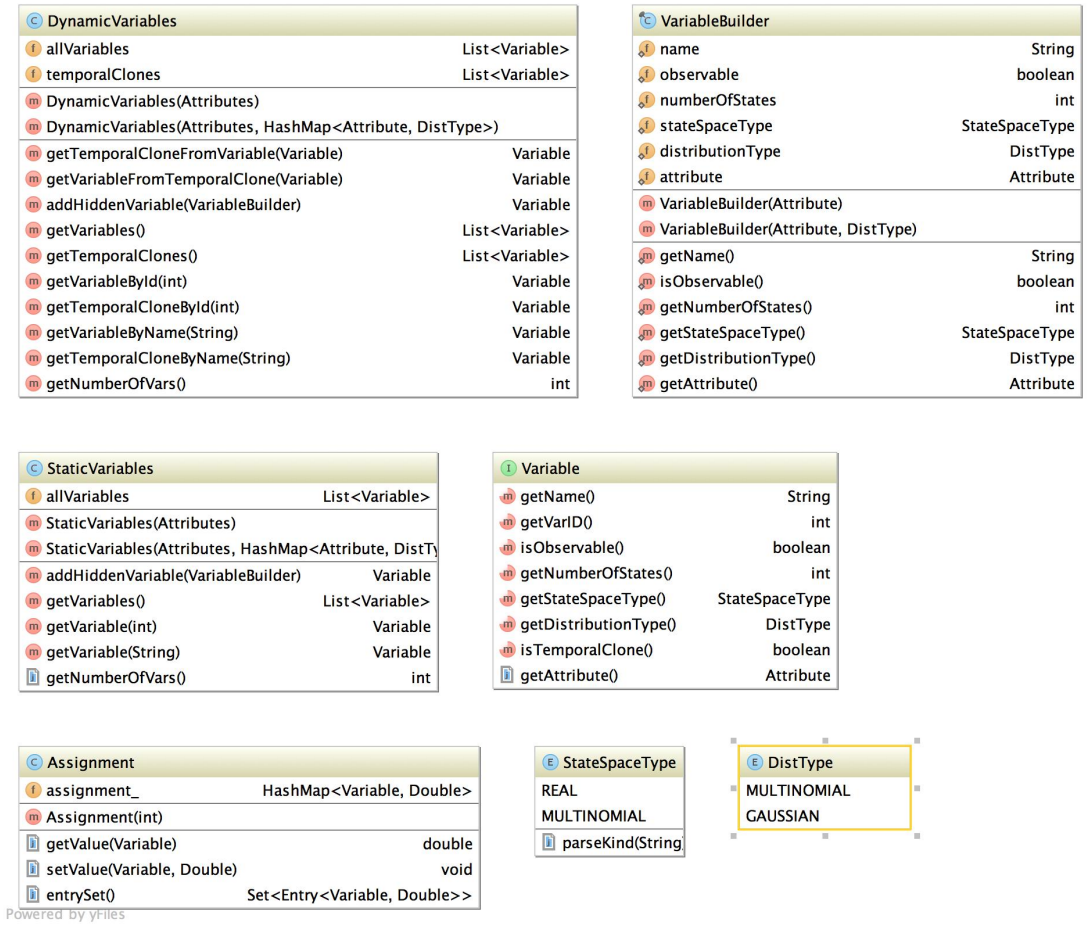


13.7 Package eu.amidst.core.database.arffWekaReader

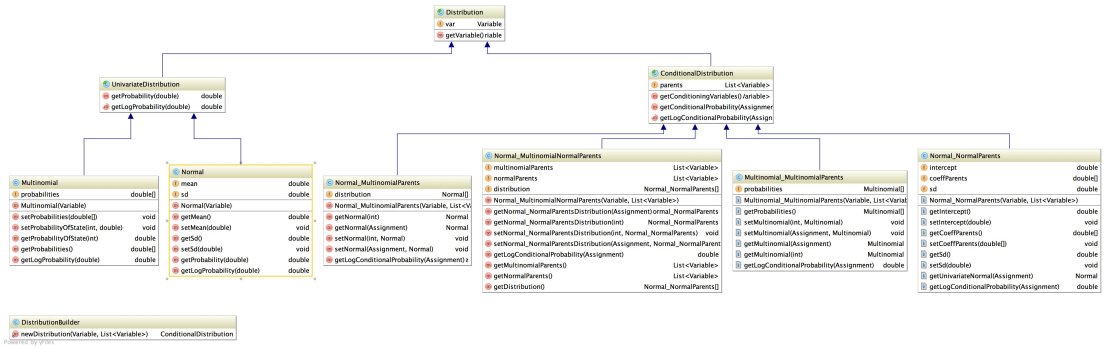


13.8 Package eu.amidst.core.variables

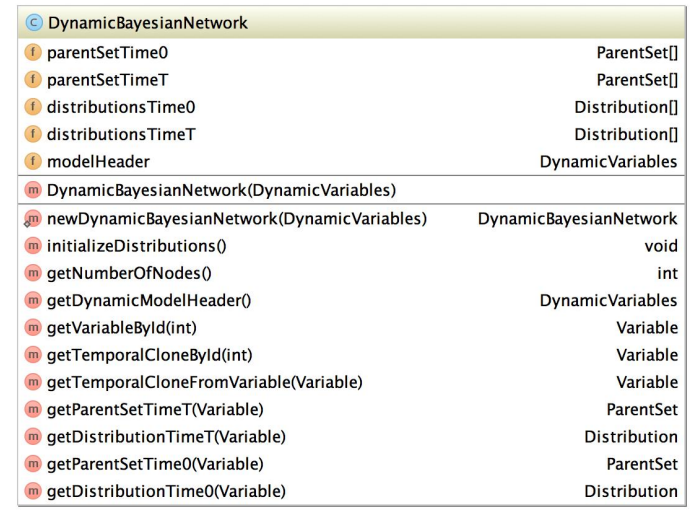
Figure 1: Class diagram of the package: `core.variables`

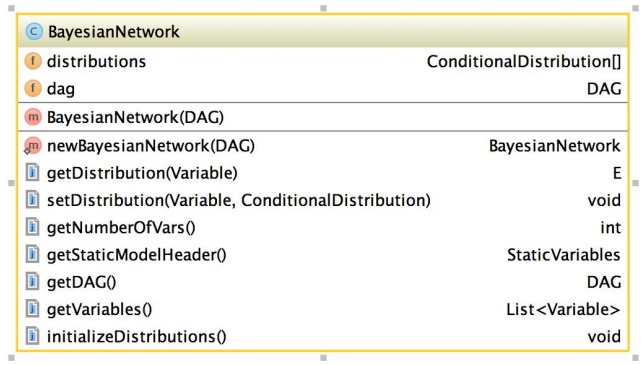


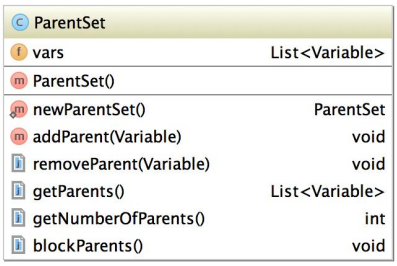
13.9 Package eu.amidst.core.distribution

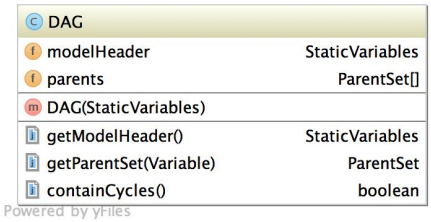


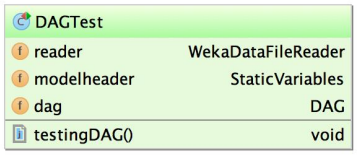
13.10 Package eu.amidst.core.modelstructure



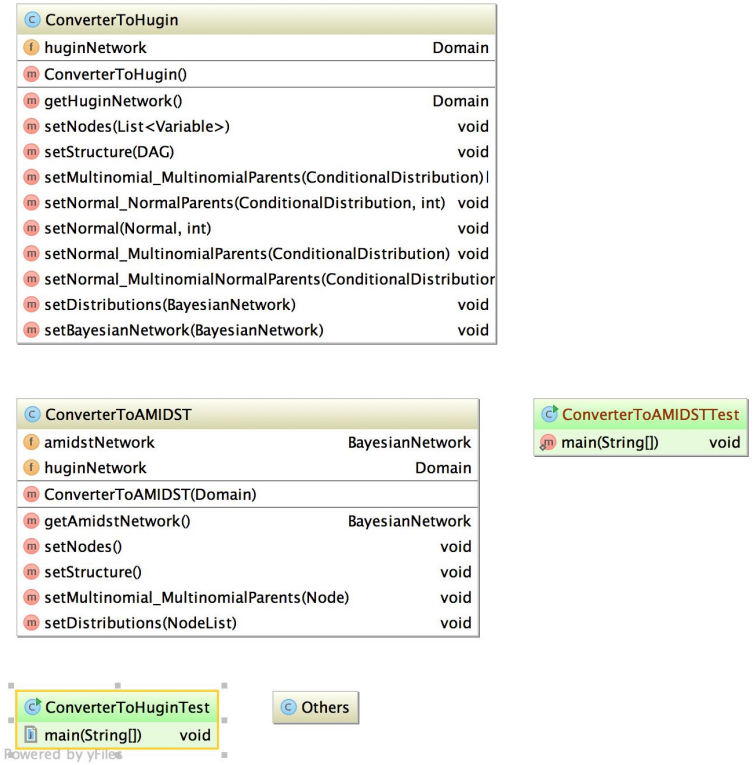






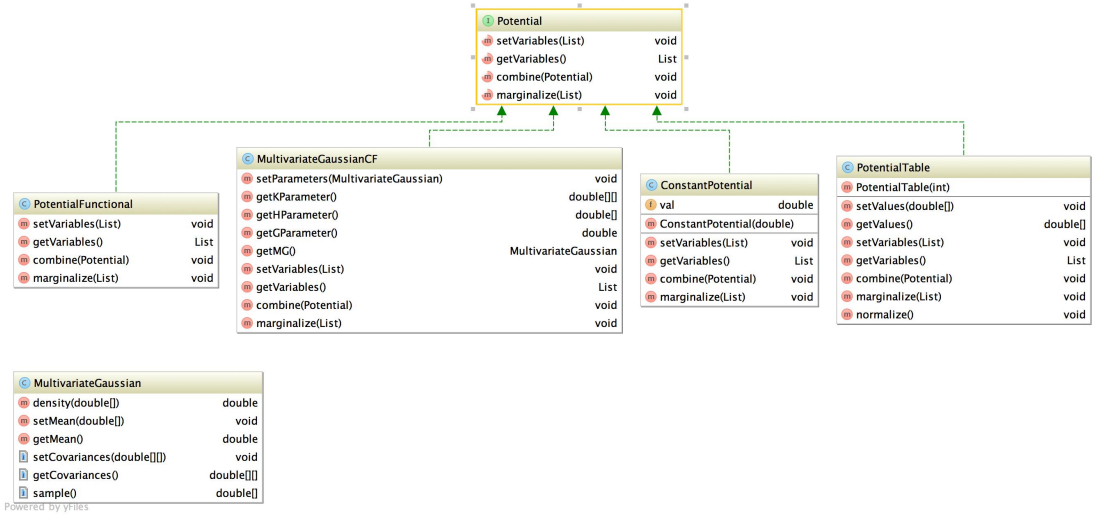


13.11 Package eu.amidst.core.huginlink

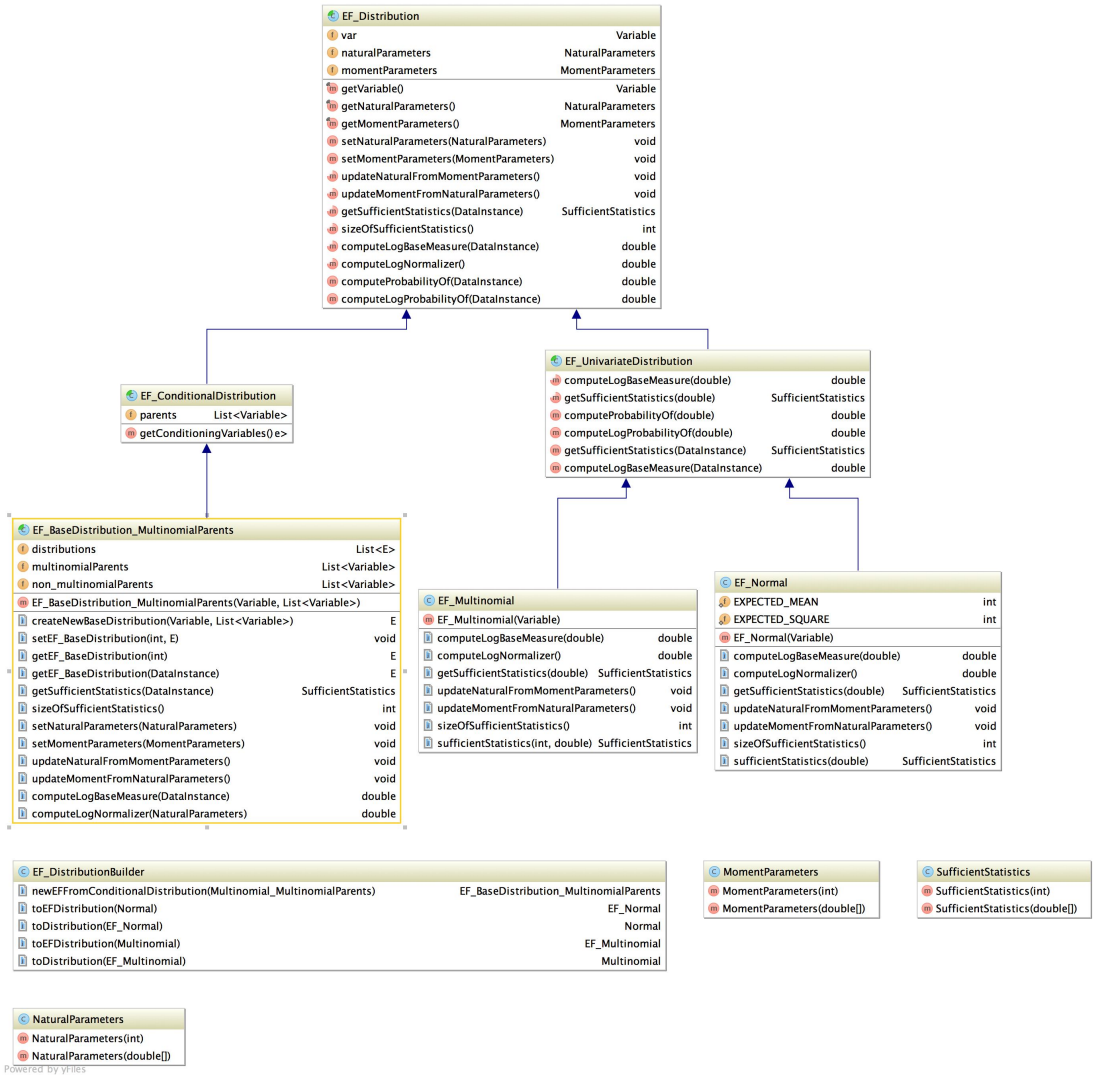


13.12 Package eu.amidst.core.potential

Figure 2: Class diagram of the package: `core.potential`



13.13 Package eu.amidst.core.exponentialfamily



13.14 Package eu.amidst.core.utils