MESSIF 3.0

Almost there...

MESSIF: General Information

- An open-source library developed at DISA
- Mainly for prototyping of indexing, search, retrieval algorithms

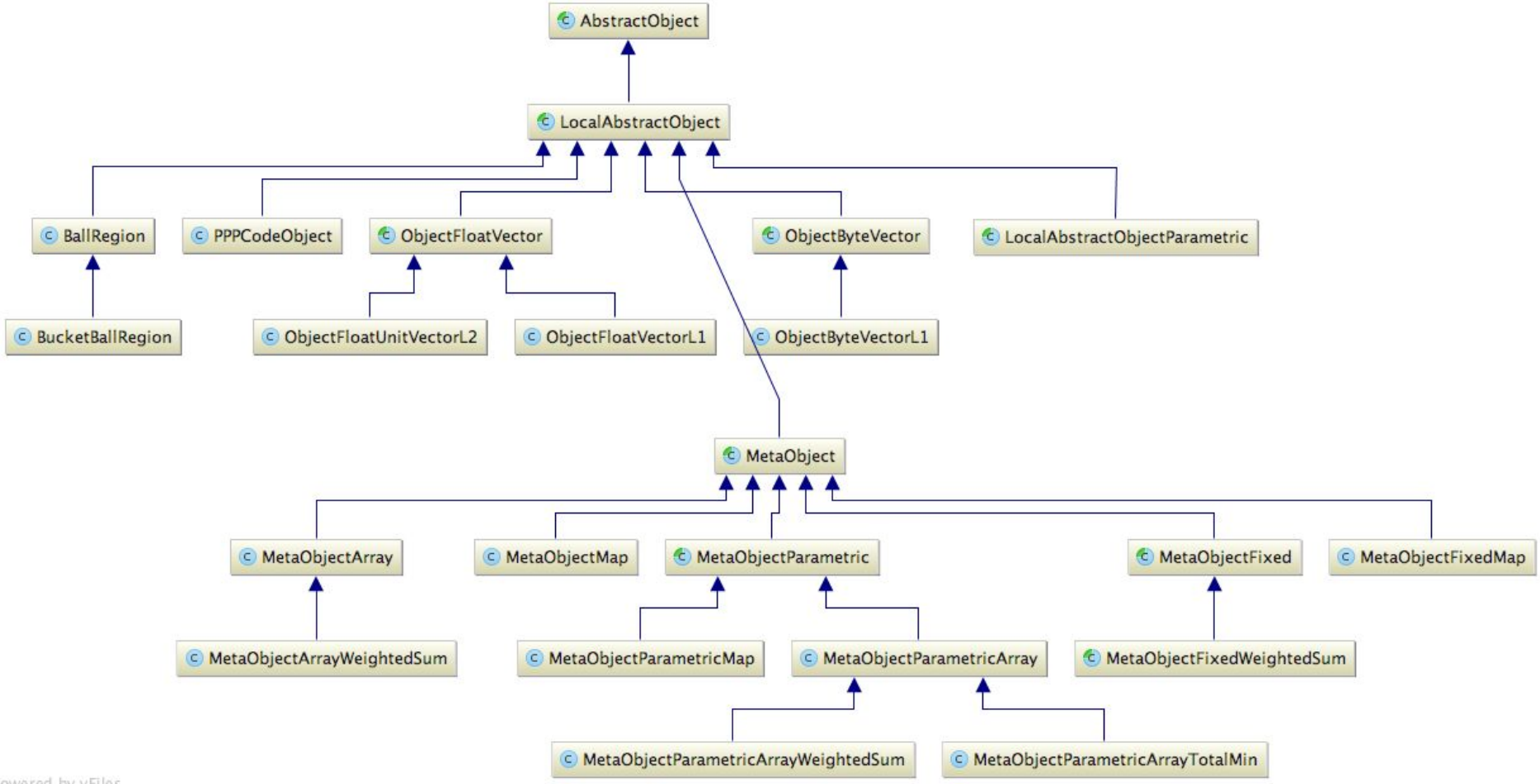
- Statistics:
  - almost 15 years
  - 460 classes
  - 84000 lines of code

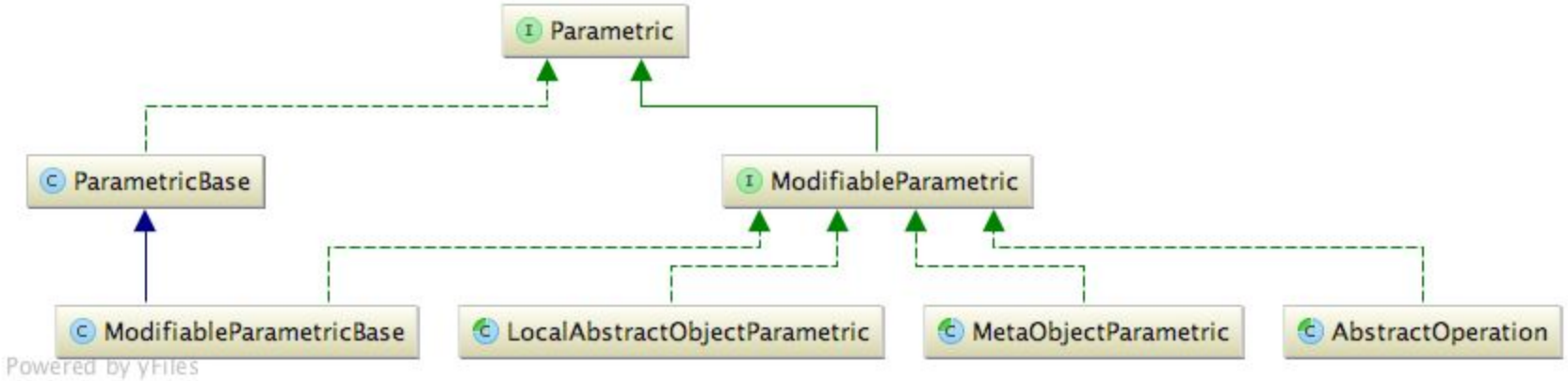
Motivations for MESSIF 3.0

- organization of code (minimize duplications)
  - e.g., separate data and distance function
- standard formats for communication
  - JSON representation of data objects and operations
- operation processing cleanup
  - separate operation, answer, processors

Data Objects

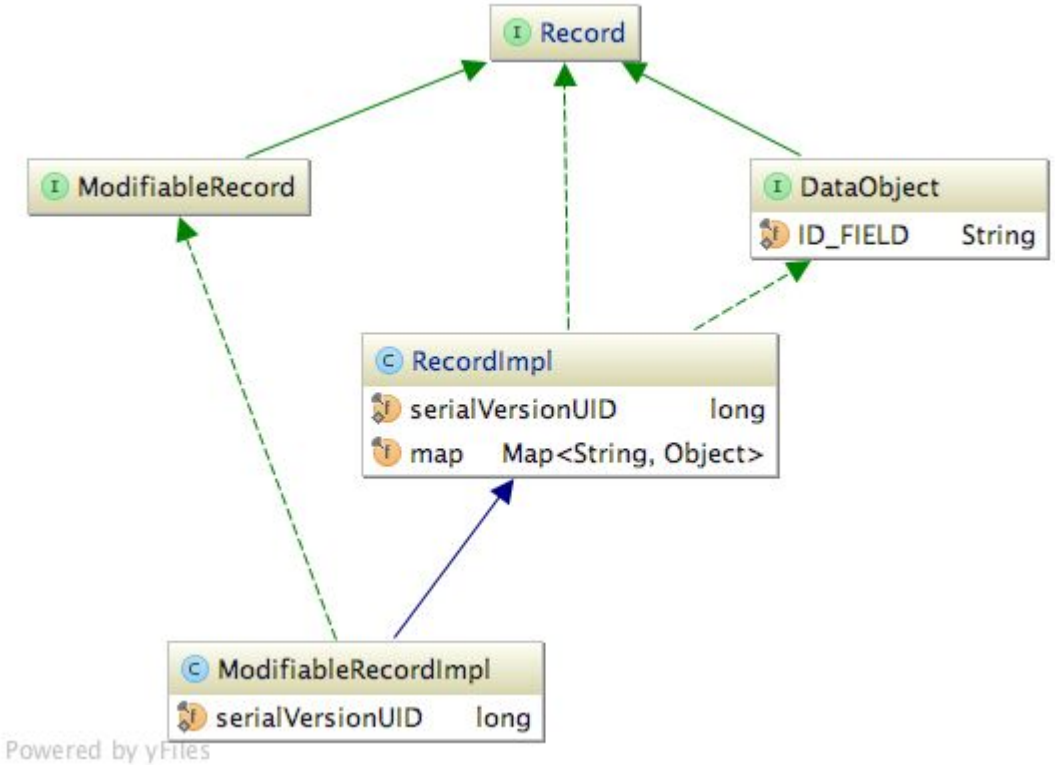
- MESSIF is tight with metric-based algorithms
  - data object + distance
- Lastly, we also work with different data types
  - keywords, sequences, simple attributes
    - distances not always natural
  - one “data object” often combines several features
    - one data collection, but various indexes & storages
- Different distance functions for the same data
  - combining & weighting distances
  - index provides candidate set, then filtering & reranking


AbstractObject
LocalAbstractObject
BallRegion
PPPCodeObject
ObjectFloatVector
ObjectByteVector
LocalAbstractObjectParametric
BucketBallRegion
ObjectFloatUnitVectorL2
ObjectFloatVectorL1
ObjectByteVectorL1
MetaObject
MetaObjectArray
MetaObjectMap
MetaObjectParametric
MetaObjectFixed
MetaObjectFixedMap
MetaObjectArrayWeightedSum
MetaObjectParametricMap
MetaObjectParametricArray
MetaObjectFixedWeightedSum
MetaObjectParametricArrayWeightedSum
MetaObjectParametricArrayTotalMin
Powered by yFiles


Parametric
ParametricBase
ModifiableParametric
ModifiableParametricBase
LocalAbstractObjectParametric
MetaObjectParametric
AbstractOperation
Powered by yFiles

Data Objects: New Design

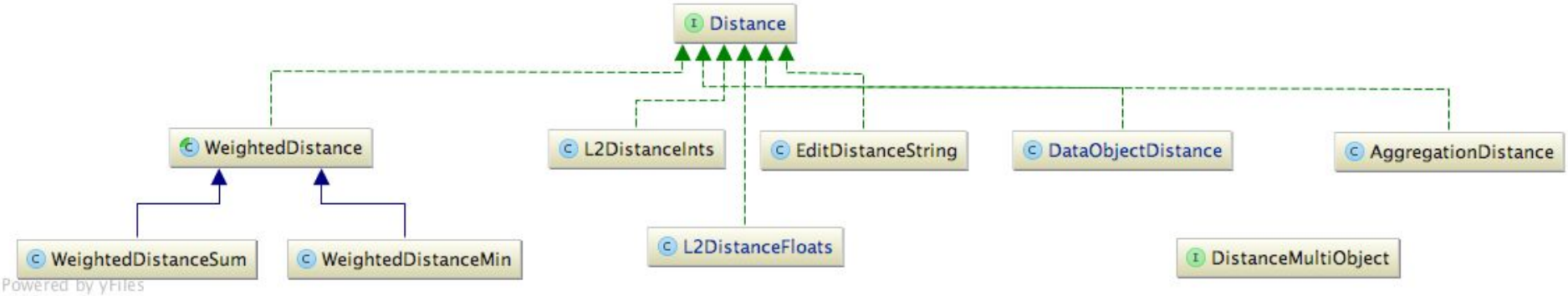
- All data objects “are JSON”
  - named fields
  - data types: int, float, String, int[], float [], String []
  - data ID (locator) - is just a field “_id”
- We need just one class: DataObject
- Distances are independent classes



Let's see the code...

Distances: New Design

- Distances work with the data



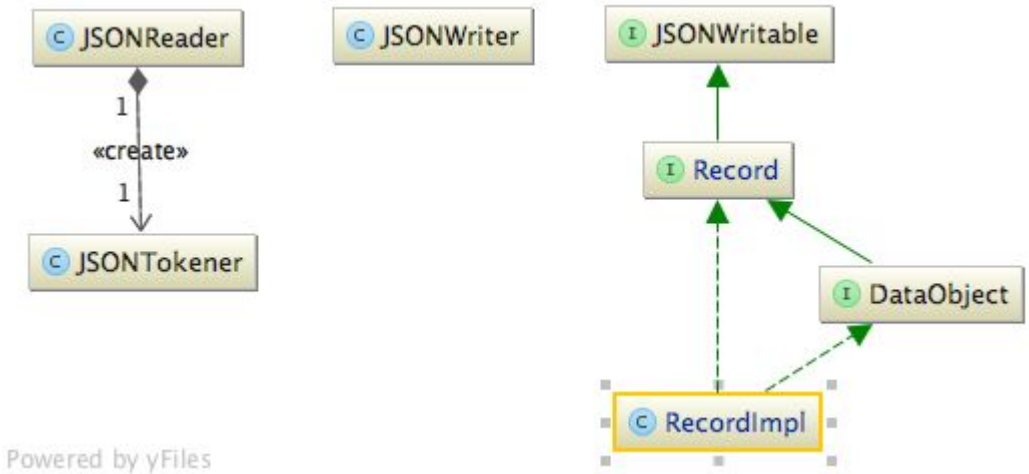
Let's see the code...

Data Objects: Reading & Writing

- MESSIF had its own format for reading/writing data into text
  - see example
- Now, we use standard JSON format
  - self-describing, flexible, less prone to errors
  - easy to use by external tools
    - standard readers/writers for all languages
  - see examples

JSON (de)serialization

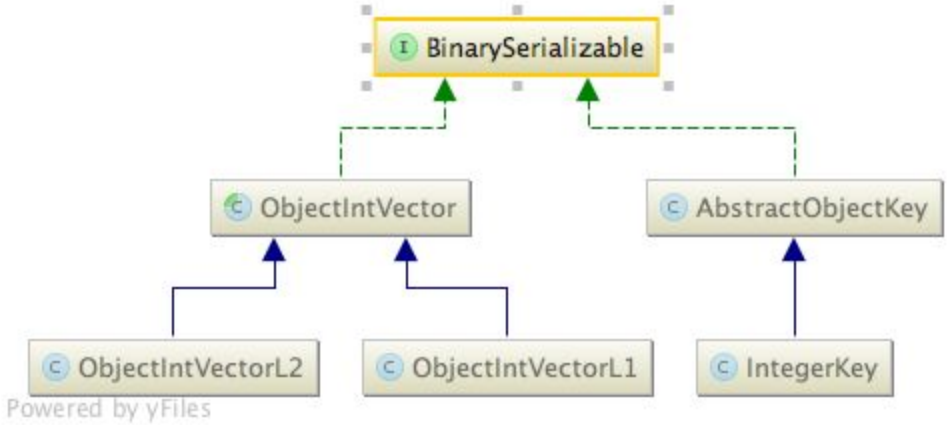
- package messif.json
  - the core taken from org.json, a free implementation

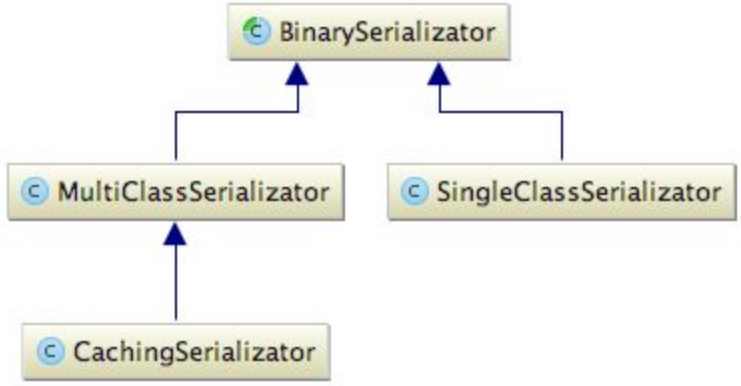


Let's see how to work with the data & distances

Binary Serialization

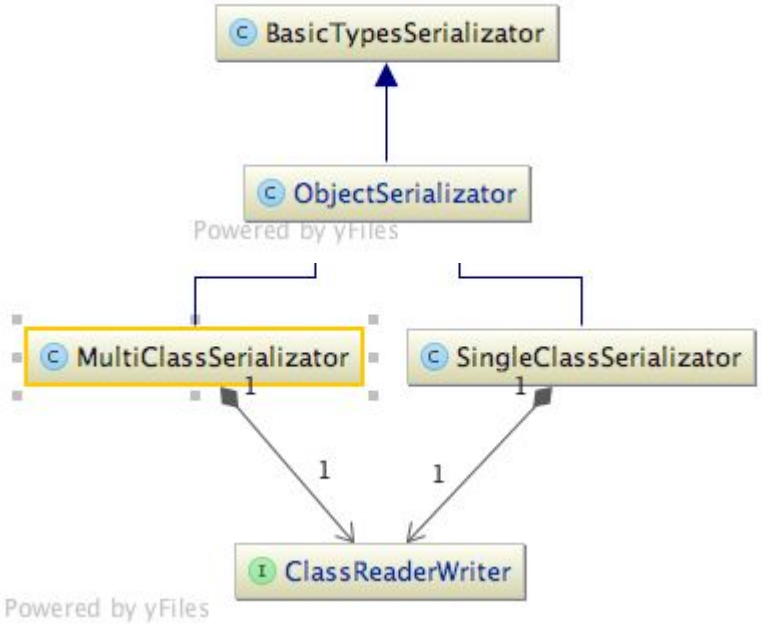
- MESSIF has its own system for efficient binary serialization (of data objects, etc.)





Binary Serialization: New Design

- Serialization of the JSON Record

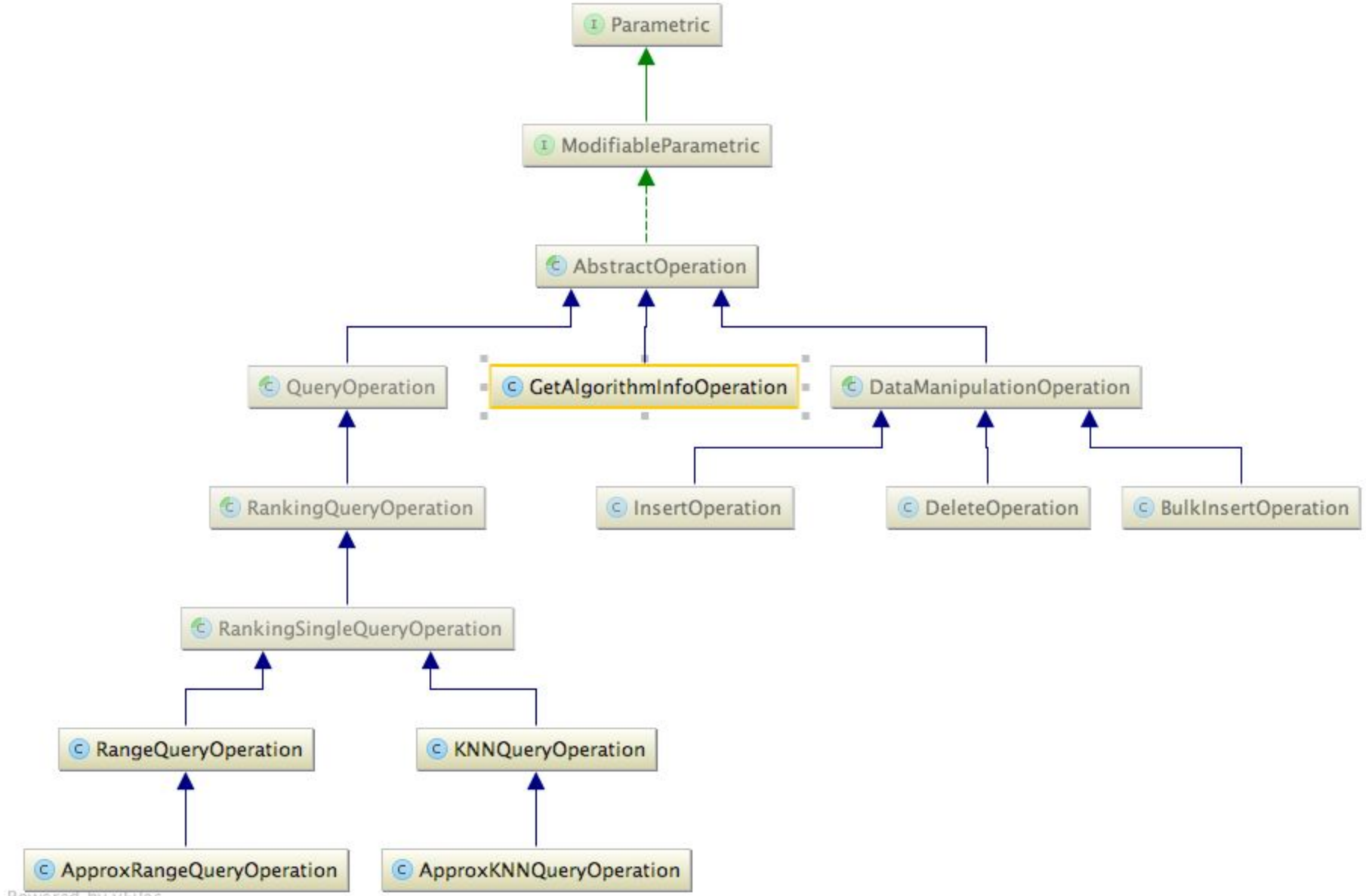


Let's have a look at the serialization and disk buckets

Operations

- Operations form the basic communication interface with a MESSIF “algorithm”
  - update data (insert, delete)
  - search: kNN, range, join

- Operations are POJO classes


Parametric
ModifiableParametric
AbstractOperation
QueryOperation
GetAlgorithmInfoOperation
DataManipulationOperation
RankingQueryOperation
InsertOperation
DeleteOperation
BulkInsertOperation
RankingSingleQueryOperation
RangeQueryOperation
KNNQueryOperation
ApproxRangeQueryOperation
ApproxKNNQueryOperation
Powered By yFiles

Operations (cont.)

- Operation classes contain:
  - input parameters (given in operation constructor)
  - answer (output)
  - additional parameters (in Parametric)
    - can be both input and output
  - helper methods for operation processing
    - building answer

Operations: Goals of New Design

- separate
  - operations (input parameters, immutable)
  - answers (output params)
  - operation processing methods
- use the flexibility of Record (named fields)
  - easy adding index-dependent parameters
- ...but keep the power of
  - checking mandatory parameters (and their types)
  - and convenient access to parameter fields

Operations: New Design

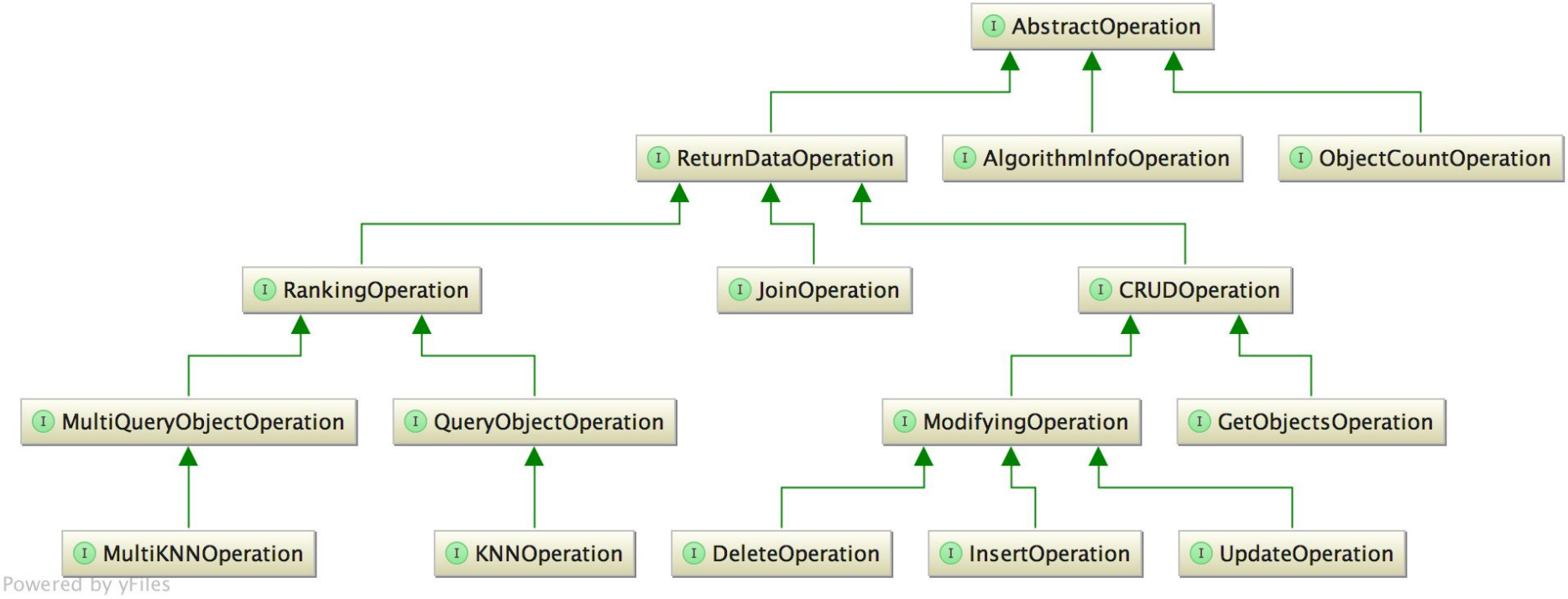
- Operation parameters are stored in a Record
- *Operations* are *interfaces*
- We exploit java.lang.reflect.Proxy
  - to wrap Record as a given interface
  - e.g. method

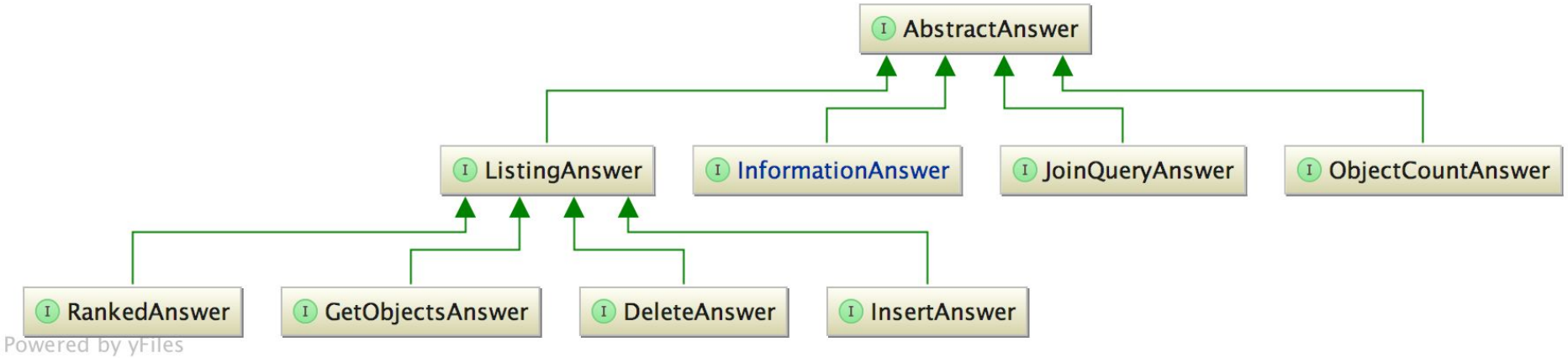
    `float RangeOperation.getRange()`

    is internally translated to

`Float Record.getRequiredField("range",Float.class)`

See example of an operation and its usage


AbstractOperation
ReturnDataOperation
AlgorithmInfoOperation
ObjectCountOperation
RankingOperation
JoinOperation
CRUDOperation
MultiQueryObjectOperation
QueryObjectOperation
ModifyingOperation
GetObjectsOperation
MultiKNNOperation
KNNOperation
DeleteOperation
InsertOperation
UpdateOperation
Powered by yFiles


AbstractAnswer
ListingAnswer
InformationAnswer
JoinQueryAnswer
ObjectCountAnswer
RankedAnswer
GetObjectsAnswer
DeleteAnswer
InsertAnswer
Powered by yFiles

Operation Processing

- In MESSIF 2.0, operations could have been executed on the algorithms:
  - by directly calling method using Java reflection

```
MyAlgorithm.myMethod(KNNQueryOperation op)
```

  - or via NavigationProcessor & NavigationDirectory
    - for given operation, algorithm creates a new *object* of type *NavigationProcessor*
    - and the processor is used step-by-step

Operation Processing: New Design

- Always use the concept of Processors
- Operations provide certain “helpers” that are used by the specific processors
- One can execute the operation on algorithm
  - Really as the wrapper class (InsertOperation)
  - or as a Record with type specified in field “_type”

- Algorithm has support for matching processors on operations

Application: Writing Config Files

- MESSIF Application
  - manage one or more running algorithms
  - allows to write a text configuration file
    - to create and execute operations
    - print the answer and statistics, etc.

- This concept is the same
  - the only change: creation of operations

See an example

TODO

- Remove “extractors”
  - and replace them by RecordProcessors
- AlgorithmObject: DataObject + Filters
- Management of “subdistances”

Where & How

- MESSIF is in GIT: https://bitbucket.org/disalab/messif/
  - branches: master (version2), version3
- We use Maven for packaging
  - mvn install produces
    - messif-2.3.8-DEVEL.jar
    - messif-3.0.0-DEVEL.jar
- All other DISA projects are also at https://bitbucket.org/disalab/