A tool for the reverse engineering of Java object-oriented source code into UML diagrams

Dimitris Anyfantakis

Computer Science and Engineering

University of Ioannina

Overview

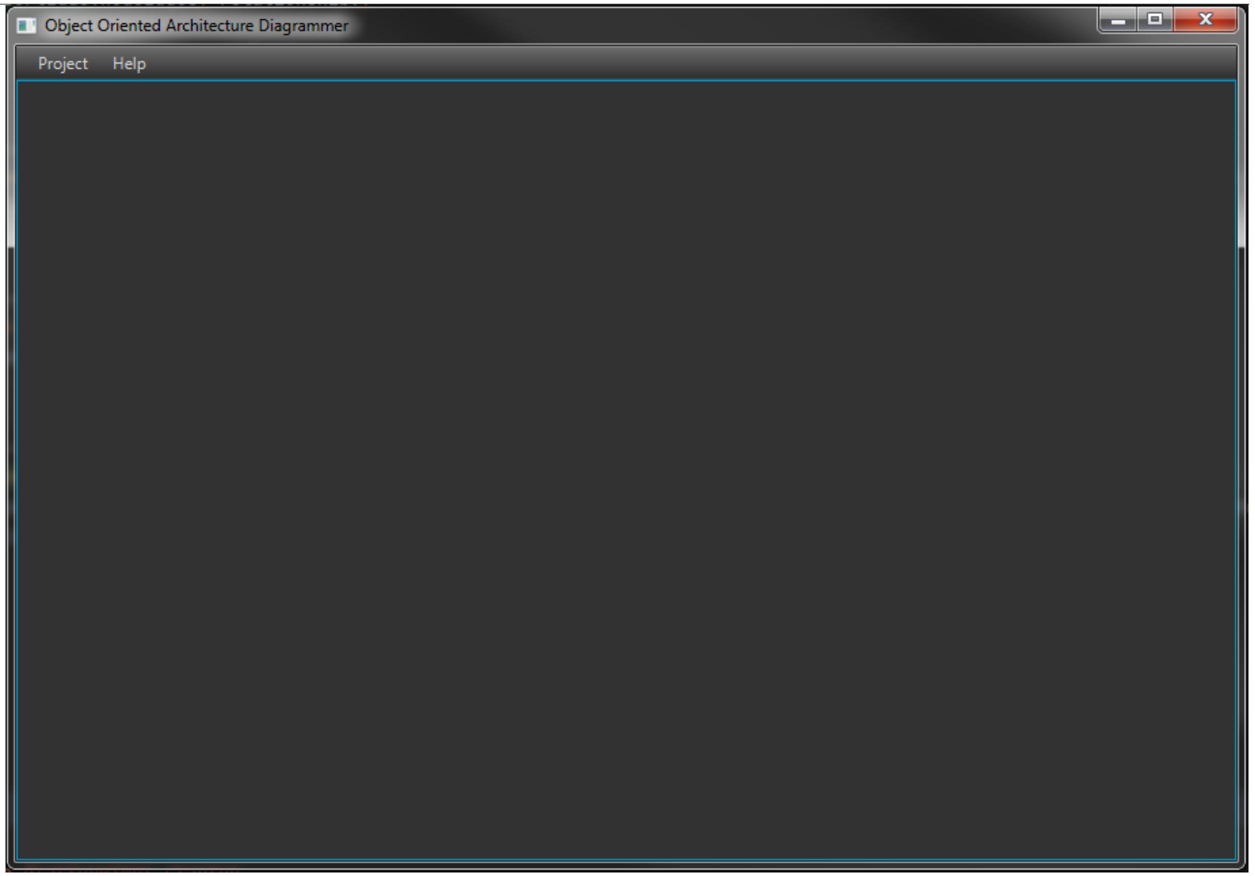
- Aim of the Thesis
- Design & Implementation
- System Validation
- Conclusion
- Future Work

Aim of the Thesis

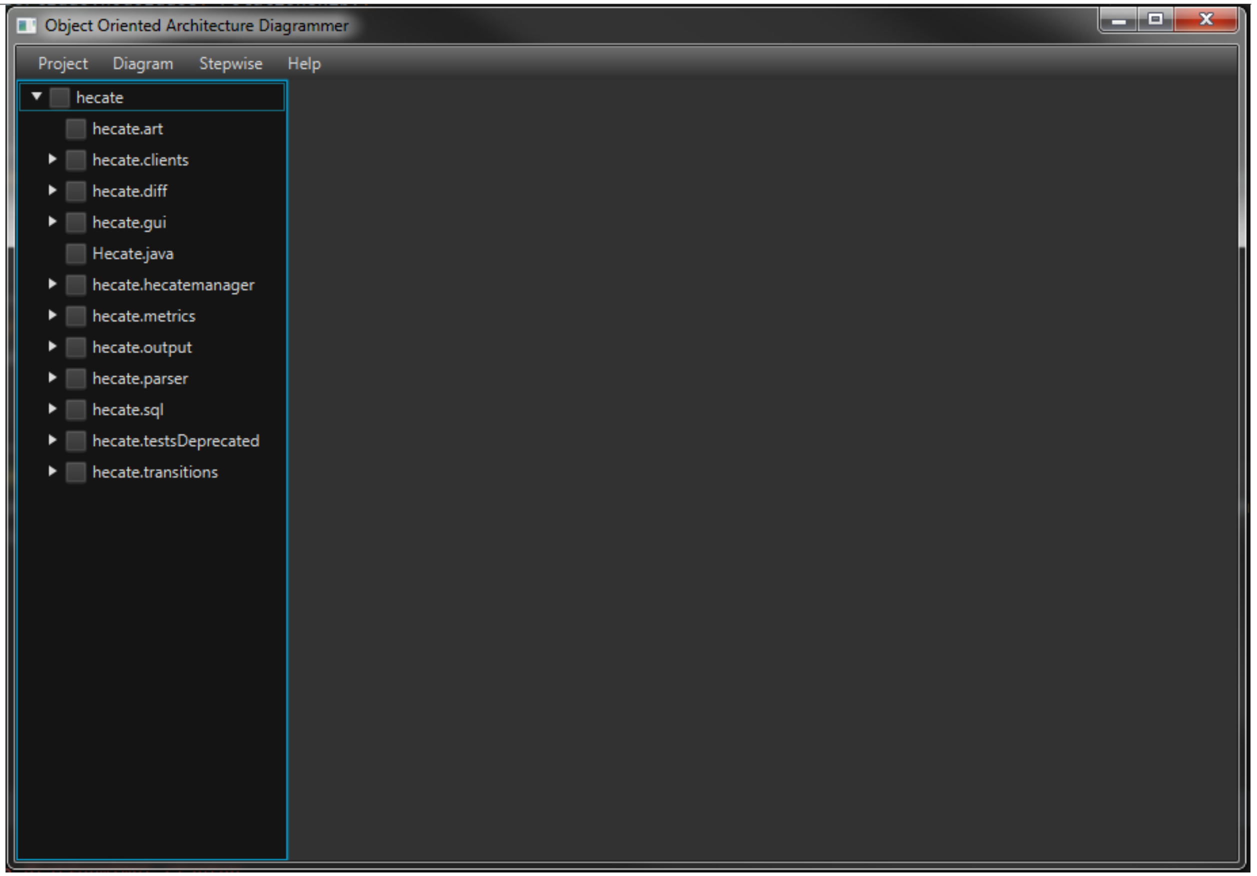
Development of a tool that produces UML diagrams by reverse engineering a Java object-oriented project.

- Parse source code using Eclipse’s Java Development Tools (JDT) Abstract Syntax Tree API
- Create a tree representing the project
- Convert the tree to a diagram
- Visualize the diagram using JavaFX
- Export the diagram in GraphML format to visualize it via yEd , text and image

Overview of the tool

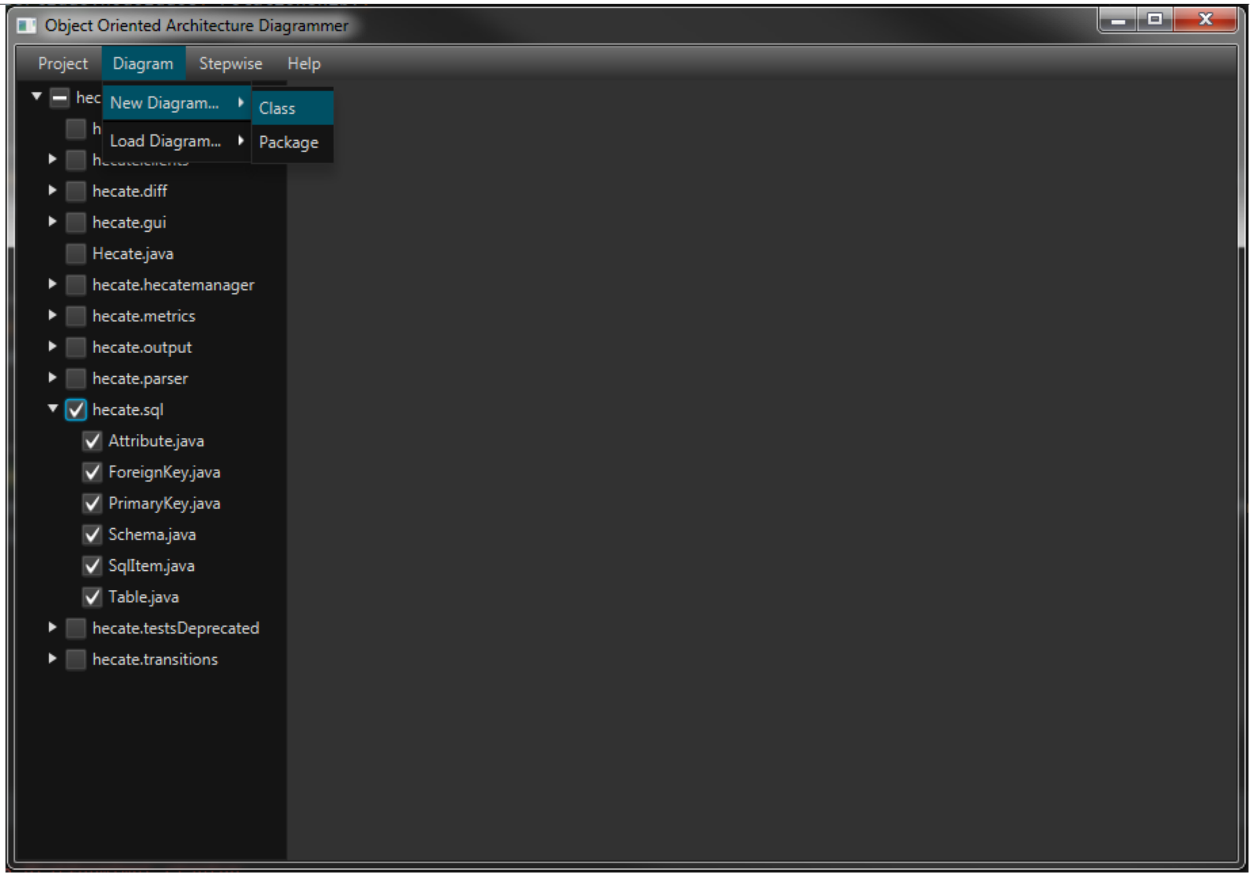


Loading a project

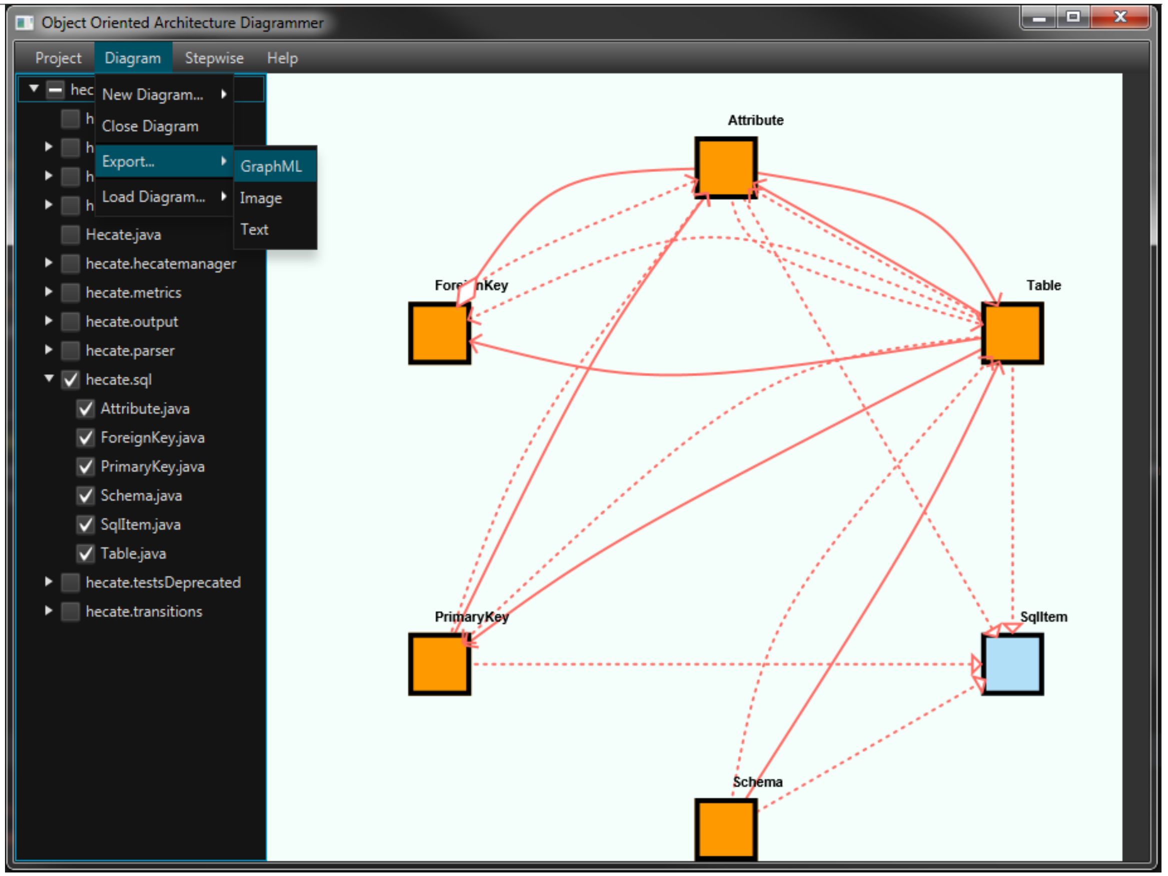


hecate
hecate.art
hecate.clients
hecate.diff
hecate.gui
Hecate.java
hecate.hecatemanager
hecate.metrics
hecate.output
hecate.parser
hecate.sql
hecate.testsDeprecated
hecate.transitions

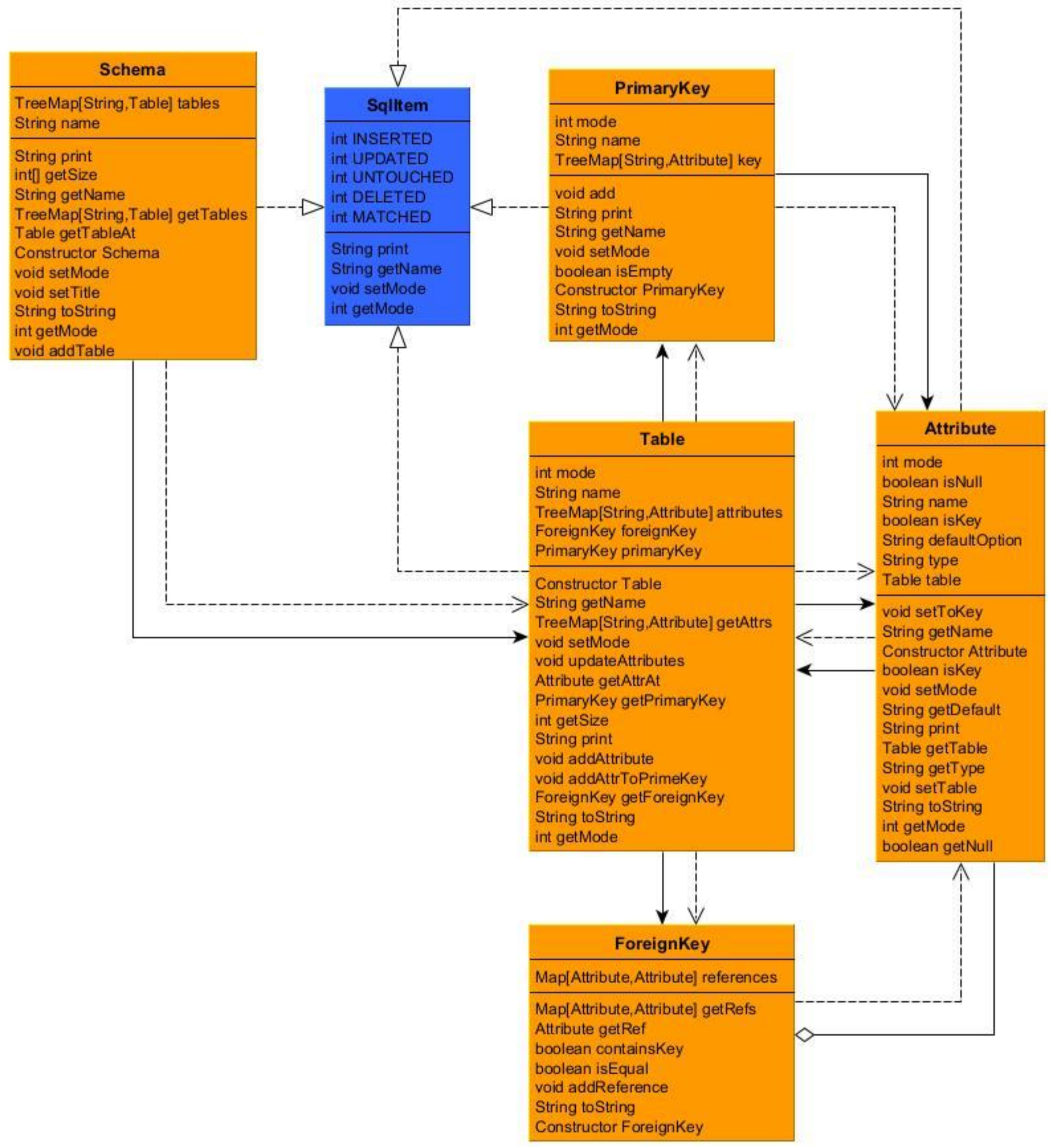
Creating a diagram



Exporting a diagram



Visualizing the exported diagram in yEd



Design & Implementation

Model

- Tree-structured software architecture to interpret the Java project
- Packages represented by nodes
- Classes & Interfaces represented by leaves
- Relationships represented by branches

Parser

- Visit project’s files using DFS
- Create nodes for every file and folder
- Create the AST of every Java source file
- Populate our tree using the information from the AST, i.e., fields’ names, types and methods’ names, parameters, return types
- Identify relationships among tree’s leaves

Diagram

- Convert the tree to a diagram
- Create a node collection for the nodes that will populate the diagram
- Create an edge collection for all relationships among the nodes
- Convert the node and edge collection into a diagram
- Map <Starting Node, Map< Ending Node, Type of relationship>>

Choosing a graph visualization library

- Extended the JavaFXSmartGraph library
    - Implemented closed, open and “diamond” arrows by extracting the Arrow superclass and created the closedArrow, openArrow and diamondArrow classes that extend it
    - Added dashed edges for dependencies and implementations
    - Node class now extends JavaFX’s Rectangle class instead of Circle

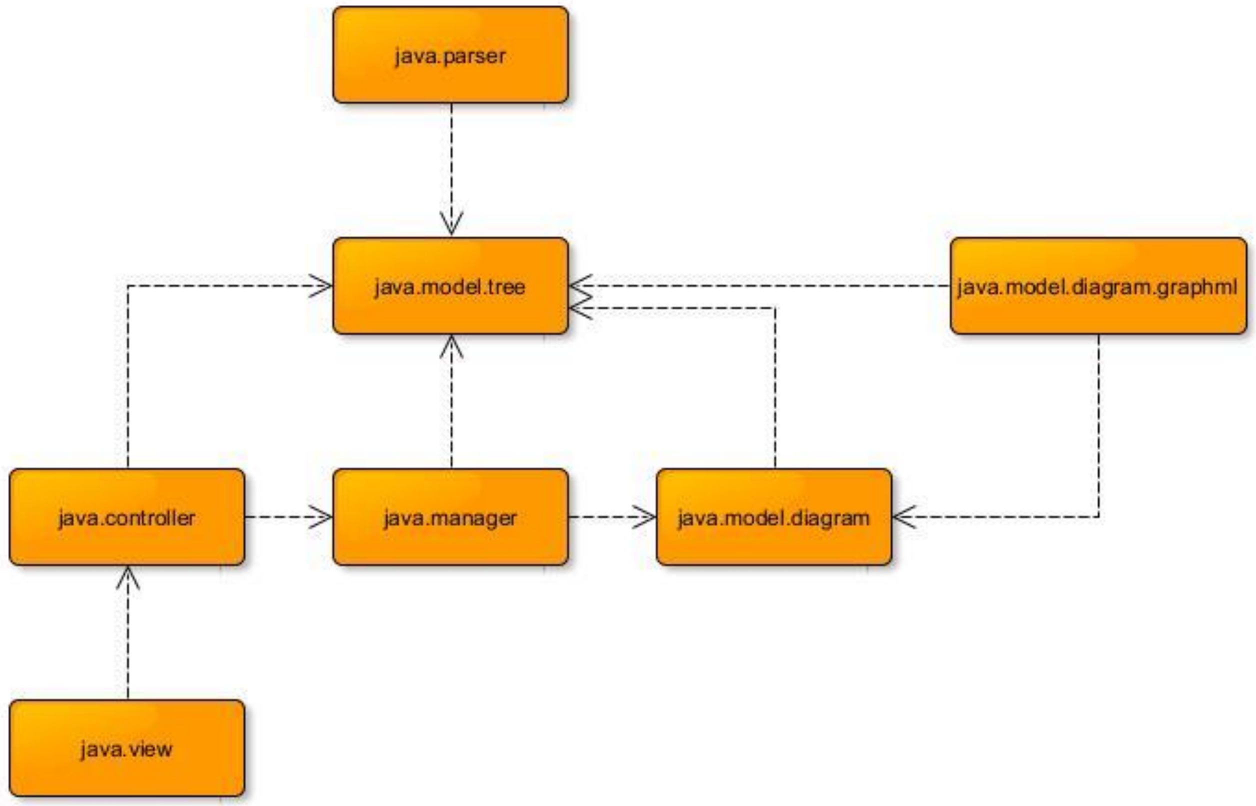
Visualizing the diagram using JavaFX

- Create a directed graph of the library
- Insert the diagram’s nodes in a graph
- Insert the diagram’s edges in the graph
- Draw the graph in a JavaFX Pane Node
- Apply the layout algorithm to the graph

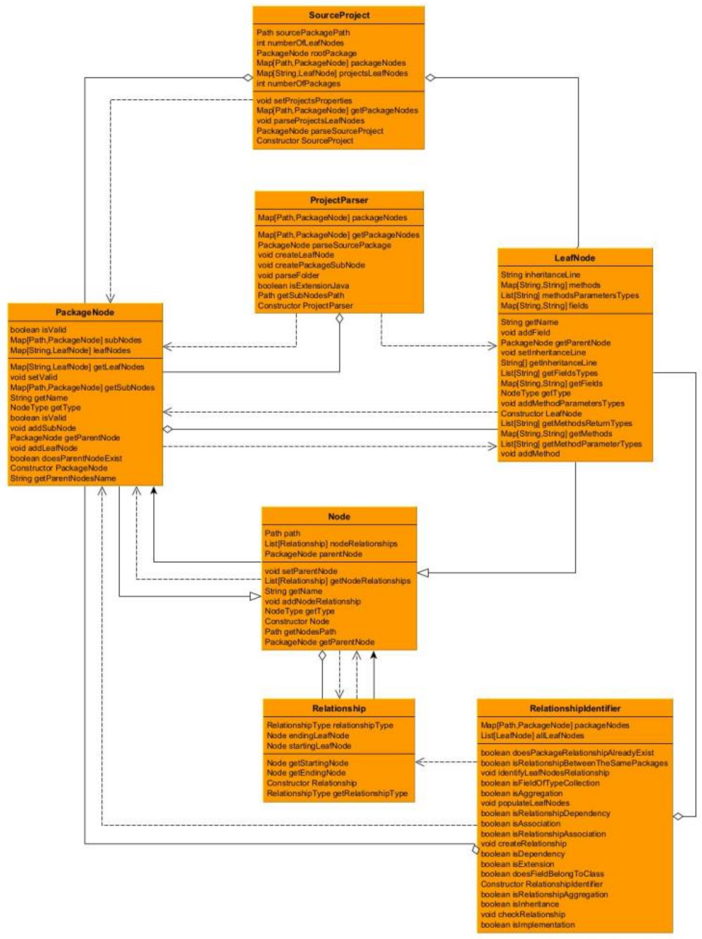
Exporting the diagram to GraphML format

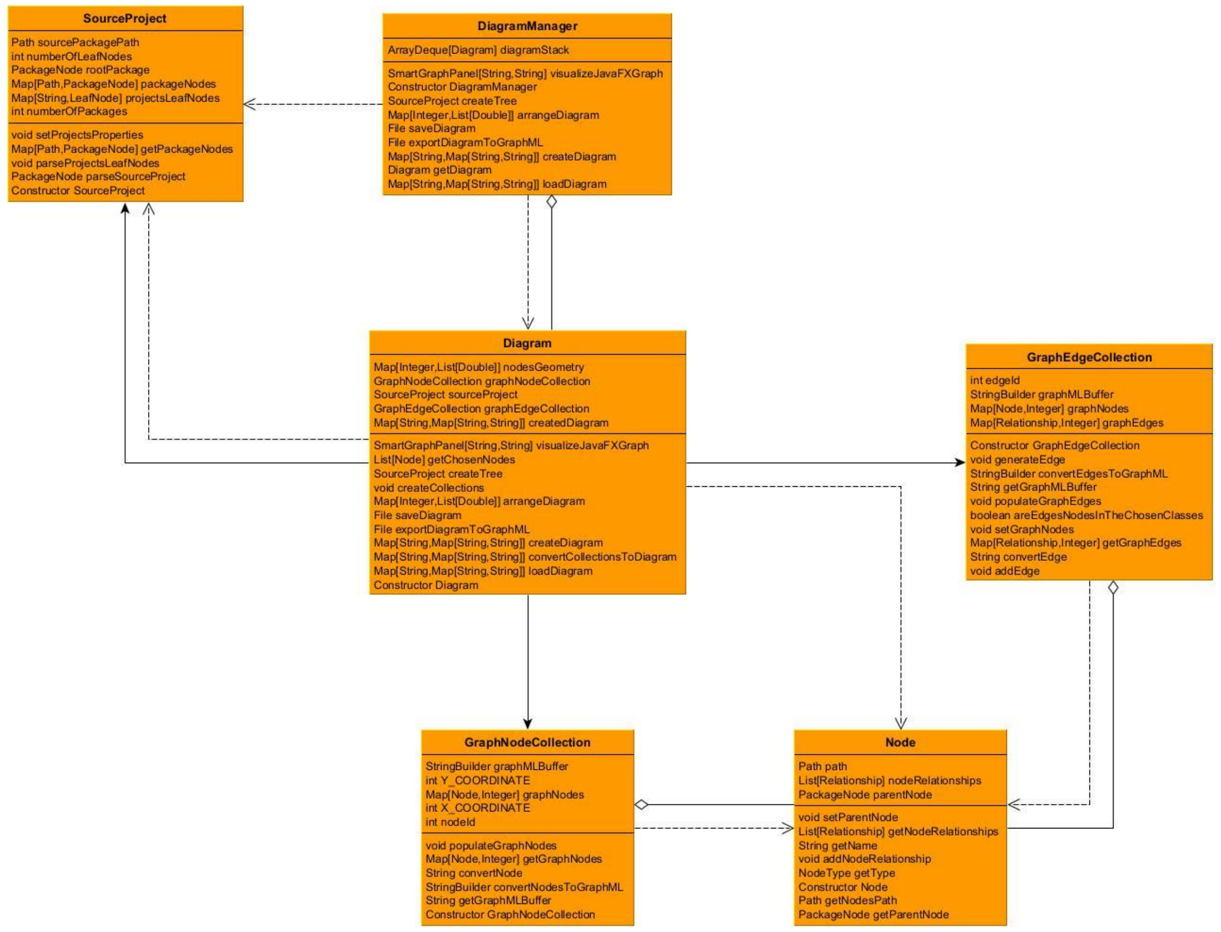
- Create a Jung graph and populate it with our diagram’s nodes and edges
- Arrange the graph using Jung’s SpringLayout algorithm
- Convert the nodes and the edges of our collections using GraphML syntax
- Write the result to disk file

Package diagram



Class diagram


SourceProject
Path sourcePackagePath
int numberOfLeafNodes
PackageNode rootPackage
Map[Path,PackageNode] packageNodes
Map[String,LeafNode] projectsLeafNodes
int numberOfPackages
void setProjectsProperties
Map[Path,PackageNode] getPackageNodes
void parseProjectsLeafNodes
PackageNode parseSourceProject
Constructor SourceProject
ProjectParser
Map[Path,PackageNode] packageNodes
Map[Path,PackageNode] getPackageNodes
PackageNode parseSourcePackage
void createLeafNode
void createPackageSubNode
void parseFolder
boolean isExtensionJava
Path getSubNodesPath
Constructor ProjectParser
LeafNode
String inheritanceLine
Map[String,String] methods
List[String] methodsParametersTypes
Map[String,String] fields
String getName
void addField
PackageNode getParentNode
void setInheritanceLine
String[] getInheritanceLine
List[String] getFieldsTypes
Map[String,String] getFields
NodeType getType
void addMethodParametersTypes
Constructor LeafNode
List[String] getMethodsReturnTypes
Map[String,String] getMethods
List[String] getMethodParameterTypes
void addMethod
PackageNode
boolean isValid
Map[Path,PackageNode] subNodes
Map[String,LeafNode] leafNodes
Map[String,LeafNode] getLeafNodes
void setValid
Map[Path,PackageNode] getSubNodes
String getName
NodeType getType
boolean isValid
void addSubNode
PackageNode getParentNode
void addLeafNode
boolean doesParentNodeExist
Constructor PackageNode
String getParentNodesName
Node
Path path
List[Relationship] nodeRelationships
PackageNode parentNode
void setParentNode
List[Relationship] getNodeRelationships
String getName
void addNodeRelationship
NodeType getType
Constructor Node
Path getNodesPath
PackageNode getParentNode
Relationship
RelationshipType relationshipType
Node endingLeafNode
Node startingLeafNode
Node getStartingNode
Node getEndingNode
Constructor Relationship
RelationshipType getRelationshipType
RelationshipIdentifier
Map[Path,PackageNode] packageNodes
List[LeafNode] allLeafNodes
boolean doesPackageRelationshipAlreadyExist
boolean isRelationshipBetweenTheSamePackages
void identifyLeafNodesRelationship
boolean isFieldOfTypeCollection
boolean isAggregation
void populateLeafNodes
boolean isRelationshipDependency
boolean isAssociation
boolean isRelationshipAssociation
void createRelationship
boolean isDependency
boolean isExtension
boolean doesFieldBelongToClass
Constructor RelationshipIdentifier
boolean isRelationshipAggregation
boolean isInheritance
void checkRelationship
boolean isImplementation

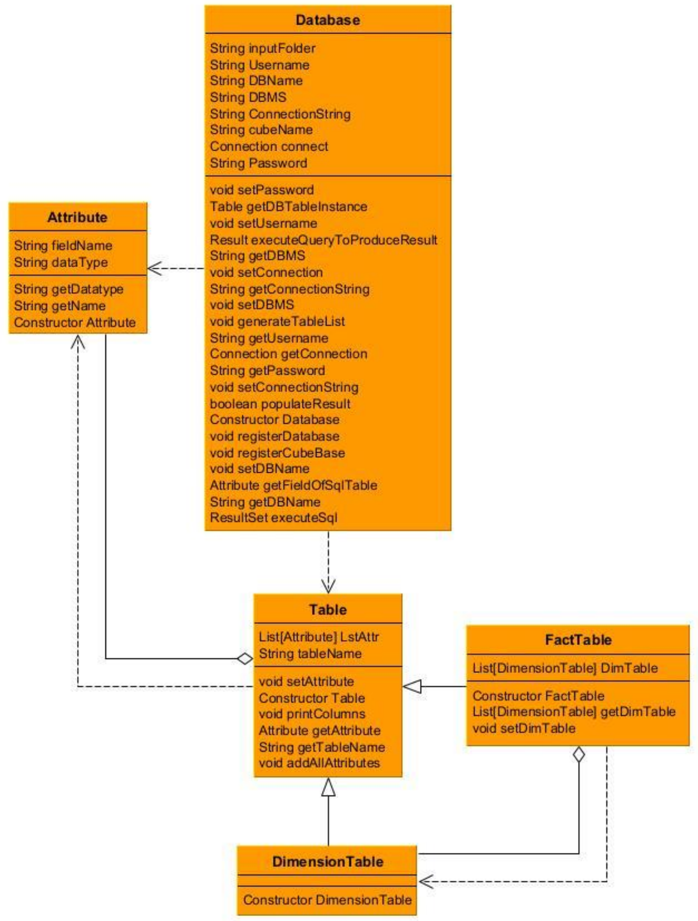

SourceProject
Path sourcePackagePath
int numberOfLeafNodes
PackageNode rootPackage
Map[Path,PackageNode] packageNodes
Map[String,LeafNode] projectsLeafNodes
int numberOfPackages
void setProjectsProperties
Map[Path,PackageNode] getPackageNodes
void parseProjectsLeafNodes
PackageNode parseSourceProject
Constructor SourceProject
DiagramManager
ArrayDeque[Diagram] diagramStack
SmartGraphPanel[String,String] visualizeJavaFXGraph
Constructor DiagramManager
SourceProject createTree
Map[Integer,List[Double]] arrangeDiagram
File saveDiagram
File exportDiagramToGraphML
Map[String,Map[String,String]] createDiagram
Diagram getDiagram
Map[String,Map[String,String]] loadDiagram
Diagram
Map[Integer,List[Double]] nodesGeometry
GraphNodeCollection graphNodeCollection
SourceProject sourceProject
GraphEdgeCollection graphEdgeCollection
Map[String,Map[String,String]] createdDiagram
SmartGraphPanel[String,String] visualizeJavaFXGraph
List[Node] getChosenNodes
SourceProject createTree
void createCollections
Map[Integer,List[Double]] arrangeDiagram
File saveDiagram
File exportDiagramToGraphML
Map[String,Map[String,String]] createDiagram
Map[String,Map[String,String]] convertCollectionsToDiagram
Map[String,Map[String,String]] loadDiagram
Constructor Diagram
GraphEdgeCollection
int edgeId
StringBuilder graphMLBuffer
Map[Node,Integer] graphNodes
Map[Relationship,Integer] graphEdges
Constructor GraphEdgeCollection
void generateEdge
StringBuilder convertEdgesToGraphML
String getGraphMLBuffer
void populateGraphEdges
boolean areEdgesNodesInTheChosenClasses
void setGraphNodes
Map[Relationship,Integer] getGraphEdges
String convertEdge
void addEdge
GraphNodeCollection
StringBuilder graphMLBuffer
int Y_COORDINATE
Map[Node,Integer] graphNodes
int X_COORDINATE
int nodeId
void populateGraphNodes
Map[Node,Integer] getGraphNodes
String convertNode
StringBuilder convertNodesToGraphML
String getGraphMLBuffer
Constructor GraphNodeCollection
Node
Path path
List[Relationship] nodeRelationships
PackageNode parentNode
void setParentNode
List[Relationship] getNodeRelationships
String getName
void addNodeRelationship
NodeType getType
Constructor Node
Path getNodesPath
PackageNode getParentNode

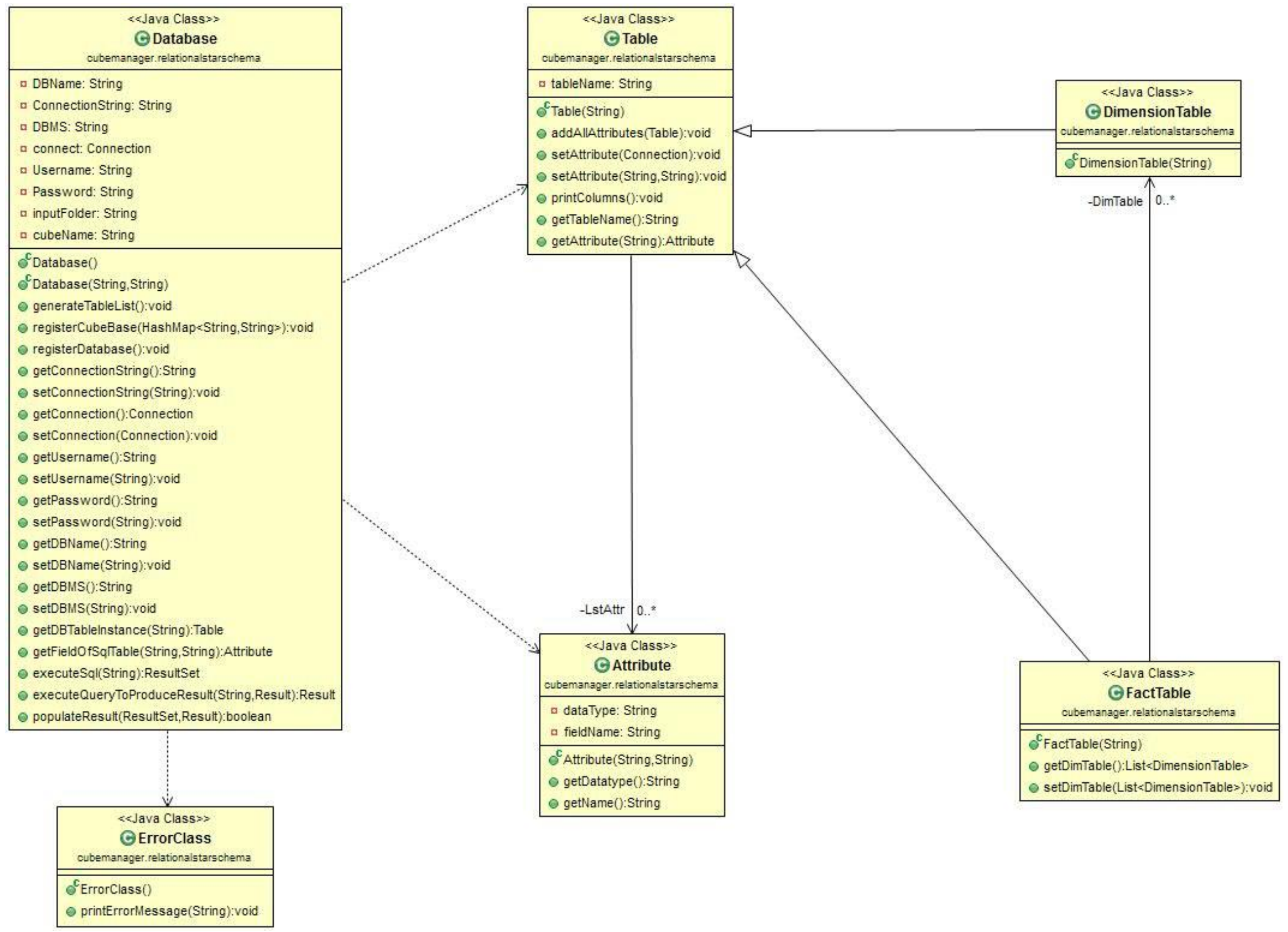
System Validation

Tests

- SourceFolderParsingTest
  - Parsing of the project
- TreeStructureArchitectureTest
  - Tree structure validation
- CollectionsDiagramConverterTest
  - Convert tree to diagram
- GraphMLConverterTest
  - Export diagram to GraphML
- JavaFXExporterTest
  - Exporting and loading of the diagram using text format
- ClassDiagramManagerTest
  - Functionalities of the diagram manager

Diagram comparison using ObjectAid

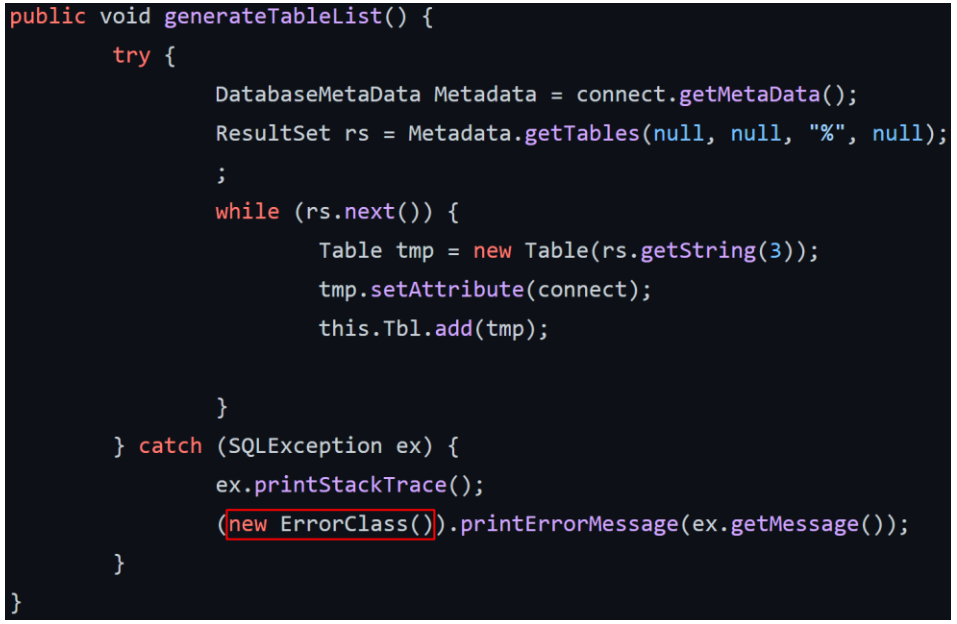



<<Java Class>>
Database
cubemanager.relationalstarschema
DBName: String
ConnectionString: String
DBMS: String
connect: Connection
Username: String
Password: String
inputFolder: String
cubeName: String
Database()
Database(String,String)
generateTableList():void
registerCubeBase(HashMap<String,String>):void
registerDatabase():void
getConnectionString():String
setConnectionString(String):void
getConnection():Connection
setConnection(Connection):void
getUsername():String
setUsername(String):void
getPassword():String
setPassword(String):void
getDBName():String
setDBName(String):void
getDBMS():String
setDBMS(String):void
getDBTableInstance(String):Table
getFieldOfSqlTable(String,String):Attribute
executeSql(String):ResultSet
executeQueryToProduceResult(String,Result):Result
populateResult(ResultSet,Result):boolean
<<Java Class>>
ErrorClass
cubemanager.relationalstarschema
ErrorClass()
printErrorMessage(String):void
<<Java Class>>
Table
cubemanager.relationalstarschema
tableName: String
Table(String)
addAllAttributes(Table):void
setAttribute(Connection):void
setAttribute(String,String):void
printColumns():void
getTableName():String
getAttribute(String):Attribute
-LstAttr
0..*
<<Java Class>>
Attribute
cubemanager.relationalstarschema
dataType: String
fieldName: String
Attribute(String,String)
getDatatype():String
getName():String
<<Java Class>>
DimensionTable
cubemanager.relationalstarschema
DimensionTable(String)
-DimTable
0..*
<<Java Class>>
FactTable
cubemanager.relationalstarschema
FactTable(String)
getDimTable():List<DimensionTable>
setDimTable(List<DimensionTable>):void

Missing arrow

- Dependency arrow that starts from Database and ends at ErrorClass class

- Instantiation of the ErrorClass without using a field

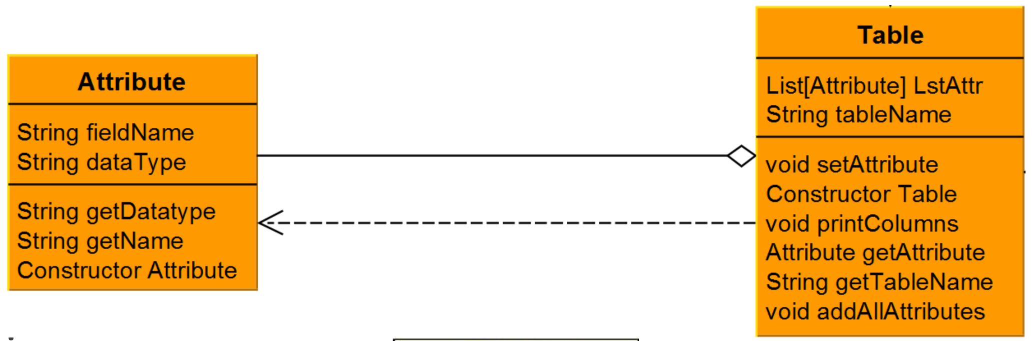


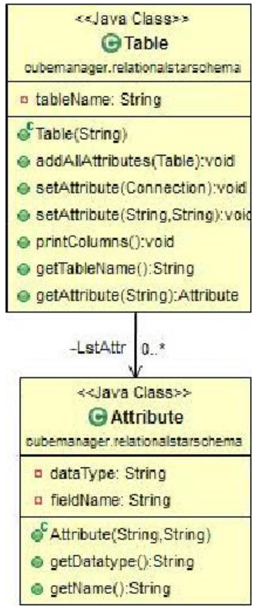
```
public void generateTableList() {
        try {
                DatabaseMetaData Metadata = connect.getMetaData();
                ResultSet rs = Metadata.getTables(null, null, "%", null);
                ;
                while (rs.next()) {
                        Table tmp = new Table(rs.getString(3));
                        tmp.setAttribute(connect);
                        this.Tbl.add(tmp);

                }
        } catch (SQLException ex) {
                ex.printStackTrace();
                (new ErrorClass()).printErrorMessage(ex.getMessage());
        }
}
```

- JDT's AST API limitation to provide information for local fields

Additional arrows

- Aggregation relationships
  - Table class has a Collection of Attribute objects
  - Similar tools like ObjectAid do not provide aggregation arrows
- Dependency relationships
  - Table depends upon the Attribute class
  - Method that returns object of Attribute





Conclusion

What does our tool offer to the designer

- Load a project and view its folder hierarchy
- Choose classes/interfaces or packages that will be included in the diagram
- Create a class or package diagram respectively
- Visualize the created diagram in our tool’s canvas
- Export the diagram in GraphML, text and image formats
- Choose different files of the same project to create a new diagram

Future Work

Parser

- Change the parsing method to improve:
  - performance
  - validity of the produced diagrams
- Use the Tree-sitter
  - Very fast parser tool
  - Parses any programming language

Visualization library

- Create visualization library that supports:
  - UML components
  - Drag and drop canvas
  - Editable canvas, i.e., deletable and moveable nodes & edges
  - Implement layout algorithm

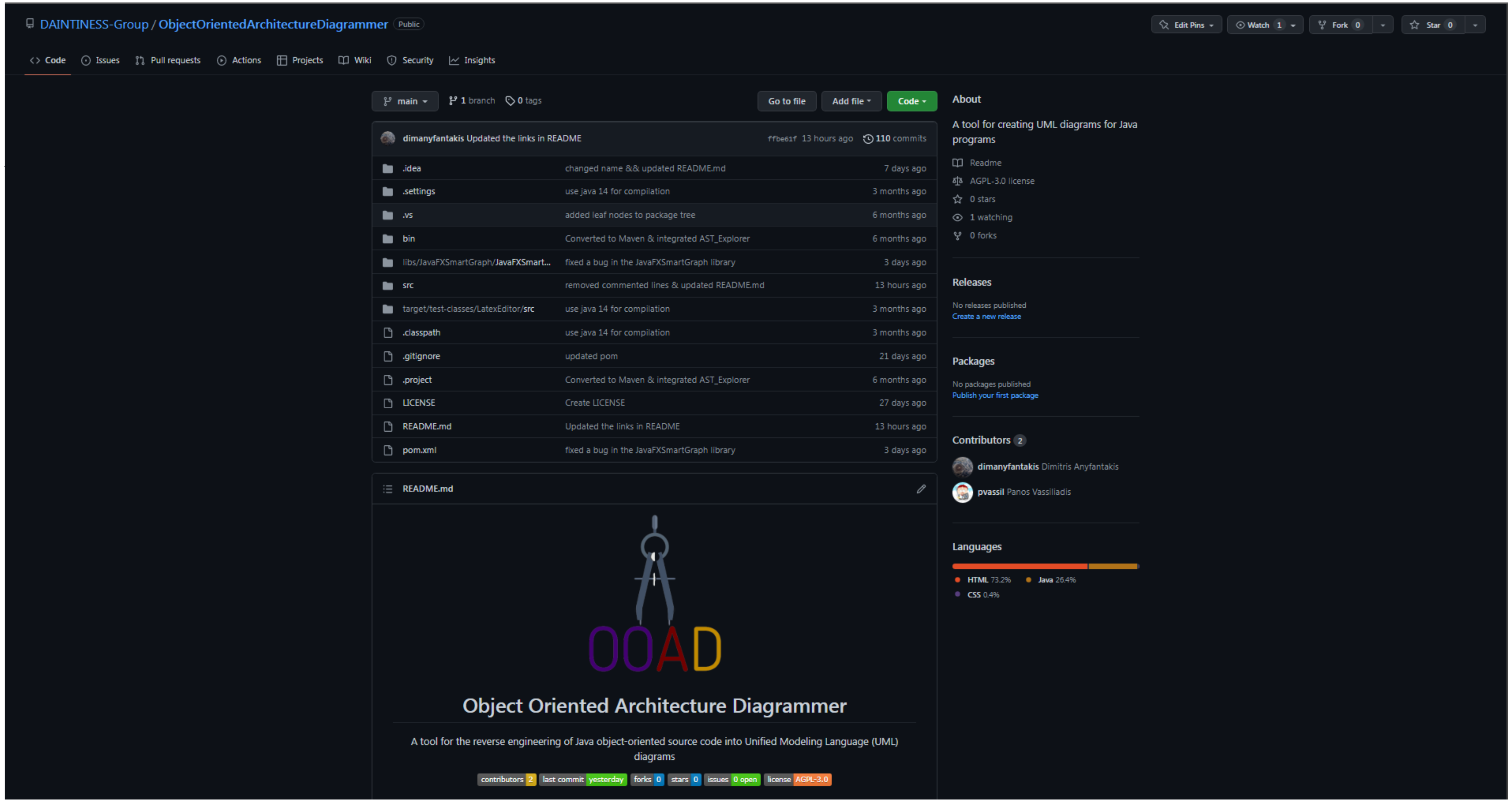
Exported diagrams

- Improve exported GraphML diagrams clarity
  - Implement an orthogonal layout algorithm
  - Use bend minimization to minimize the number of bends on the edges in the diagram.

PlantUML

- Open-source tool that uses textual descriptions to draw UML diagrams
    - Export to text file
    - Use the text file to create the image of the diagram
    - Visualize it within our tool
    - Save it as an image

Repository



dimanyfantakis Updated the links in README ffbe61f 13 hours ago 110 commits

.idea	changed name && updated README.md	7 days ago
.settings	use java 14 for compilation	3 months ago
.vs	added leaf nodes to package tree	6 months ago
bin	Converted to Maven & integrated AST_Explorer	6 months ago
libs/JavaFXSmartGraph/JavaFXSmart...	fixed a bug in the JavaFXSmartGraph library	3 days ago
src	removed commented lines & updated README.md	13 hours ago
target/test-classes/LatexEditor/src	use java 14 for compilation	3 months ago
.classpath	use java 14 for compilation	3 months ago
.gitignore	updated pom	21 days ago
.project	Converted to Maven & integrated AST_Explorer	6 months ago
LICENSE	Create LICENSE	27 days ago
README.md	Updated the links in README	13 hours ago
pom.xml	fixed a bug in the JavaFXSmartGraph library	3 days ago

README.md

OOAD

Object Oriented Architecture Diagrammer

A tool for the reverse engineering of Java object-oriented source code into Unified Modeling Language (UML) diagrams

About

A tool for creating UML diagrams for Java programs

Readme
AGPL-3.0 license
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 2

dimanyfantakis Dimitris Anyfantakis
pvassil Panos Vassiliadis

Languages

HTML 73.2% Java 26.4%
CSS 0.4%