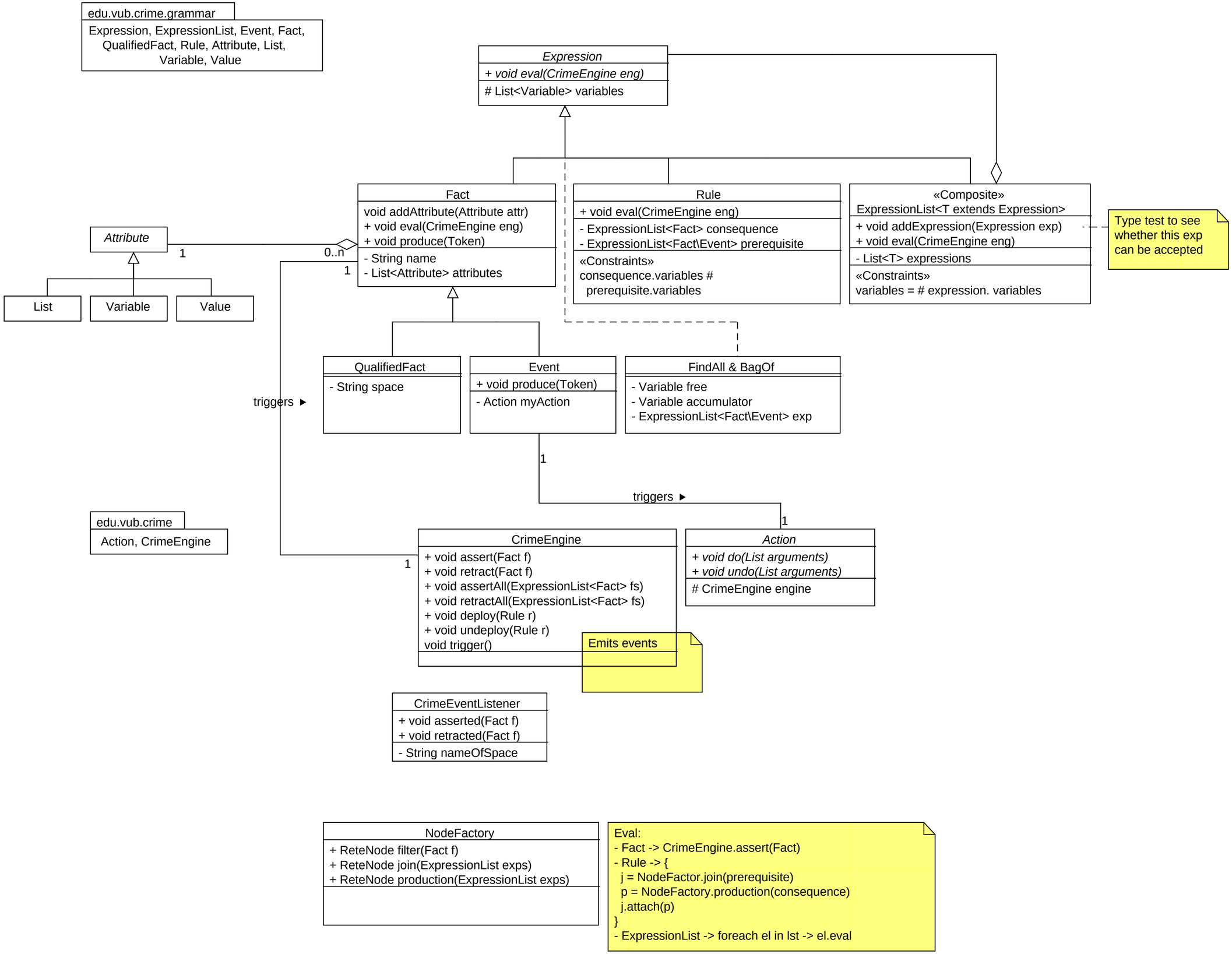
edu.vub.crime.grammar
Expression, ExpressionList, Event, Fact, QualifiedFact, Rule, Attribute, List, Variable, Value
Expression
+ void eval(CrimeEngine eng)
# List<Variable> variables
Fact
void addAttribute(Attribute attr)
+ void eval(CrimeEngine eng)
+ void produce(Token)
- String name
- List<Attribute> attributes
Rule
+ void eval(CrimeEngine eng)
- ExpressionList<Fact> consequence
- ExpressionList<Fact\Event> prerequisite
«Constraints»
consequence.variables # prerequisite.variables
«Composite»
ExpressionList<T extends Expression>
+ void addExpression(Expression exp)
+ void eval(CrimeEngine eng)
- List<T> expressions
«Constraints»
variables = # expression. variables
Type test to see whether this exp can be accepted
Attribute
List
Variable
Value
1
0..n
1
QualifiedFact
- String space
Event
+ void produce(Token)
- Action myAction
FindAll & BagOf
- Variable free
- Variable accumulator
- ExpressionList<Fact\Event> exp
triggers ▸
1
triggers ▸
1
edu.vub.crime
Action, CrimeEngine
CrimeEngine
+ void assert(Fact f)
+ void retract(Fact f)
+ void assertAll(ExpressionList<Fact> fs)
+ void retractAll(ExpressionList<Fact> fs)
+ void deploy(Rule r)
+ void undeploy(Rule r)
void trigger()
1
Action
+ void do(List arguments)
+ void undo(List arguments)
# CrimeEngine engine
Emits events
CrimeEventListener
+ void asserted(Fact f)
+ void retracted(Fact f)
- String nameOfSpace
NodeFactory
+ ReteNode filter(Fact f)
+ ReteNode join(ExpressionList exps)
+ ReteNode production(ExpressionList exps)
Eval:
- Fact -> CrimeEngine.assert(Fact)
- Rule -> {
j = NodeFactor.join(prerequisite)
p = NodeFactory.production(consequence)
j.attach(p)
}
- ExpressionList -> foreach el in lst -> el.eval