KCSE 2016 논문 발표

소프트웨어 품질관리를 위한 이종 코드 변환 프레임워크 연구

홍익대 소프트웨어공학 연구실

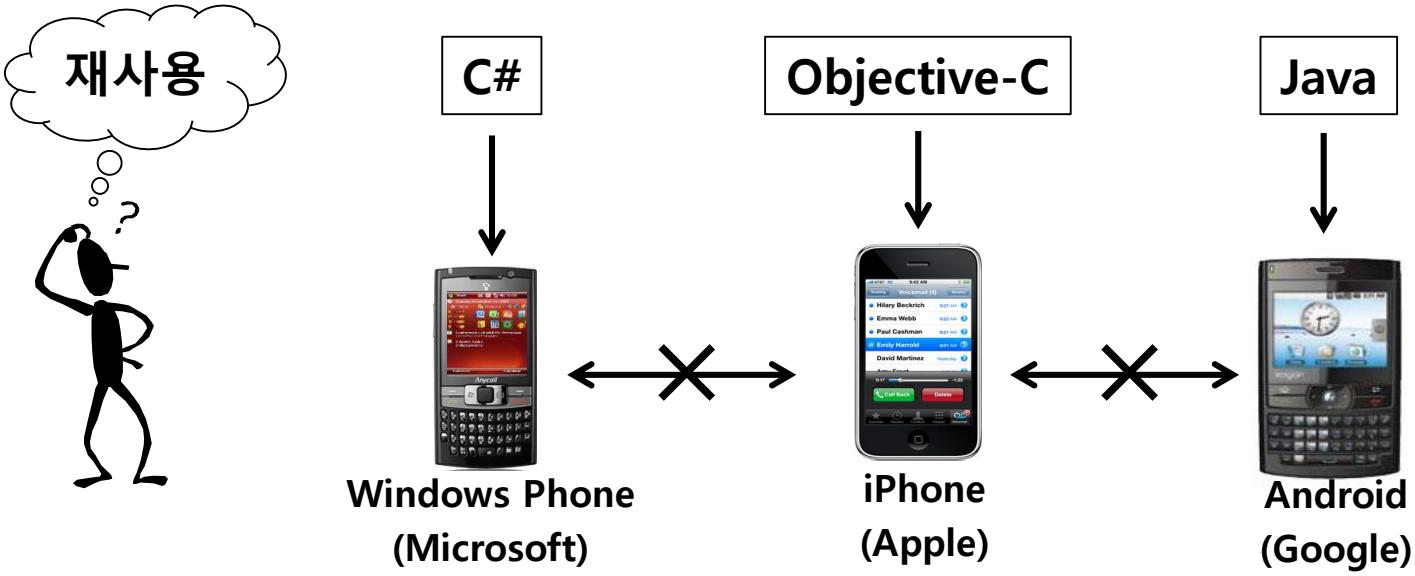
발표자 : 손현승 (son@selab.hongik.ac.kr)

발표일 : 2016. 1. 29 (금)

목 차

1. 연구 동기

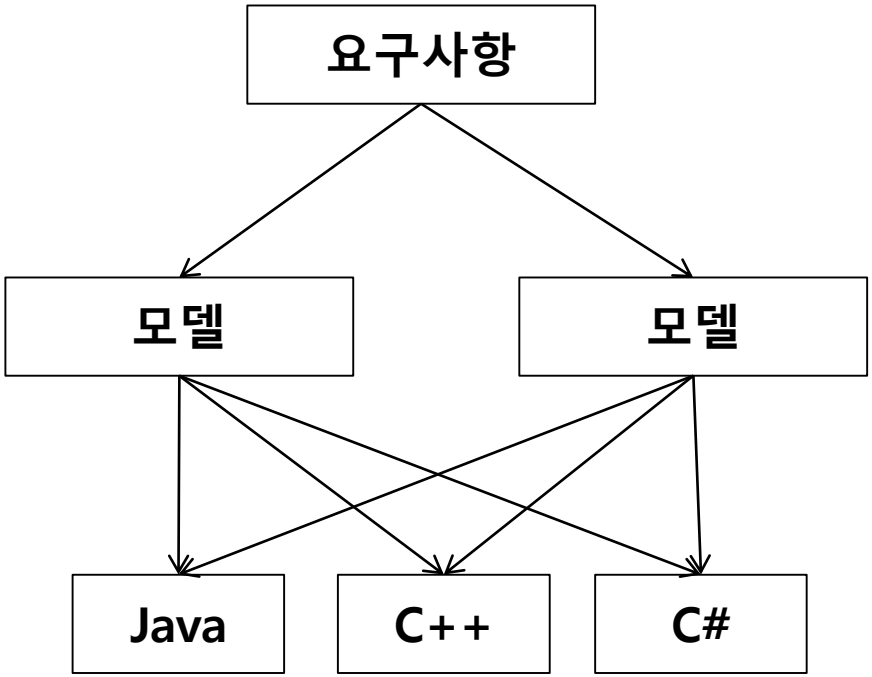
- 단일 플랫폼기반 소프트웨어
  - ✓ 플랫폼에 구현된 소프트웨어의 **재사용** 가능
  - ✓ 빠르고 안정적으로 개발가능
- 이종 플랫폼/소프트웨어 개발
  - ✓ 하나의 플랫폼으로 만들어진 소프트웨어는 플랫폼에 종속되어 다른 플랫폼에서 재사용 어려움
  - ✓ 또한, 프로그램 개발 언어가 다른 경우도 재사용이 어려움



1) 새로운 시스템, 2)기존(legacy) 시스템, 3) 연구 목표

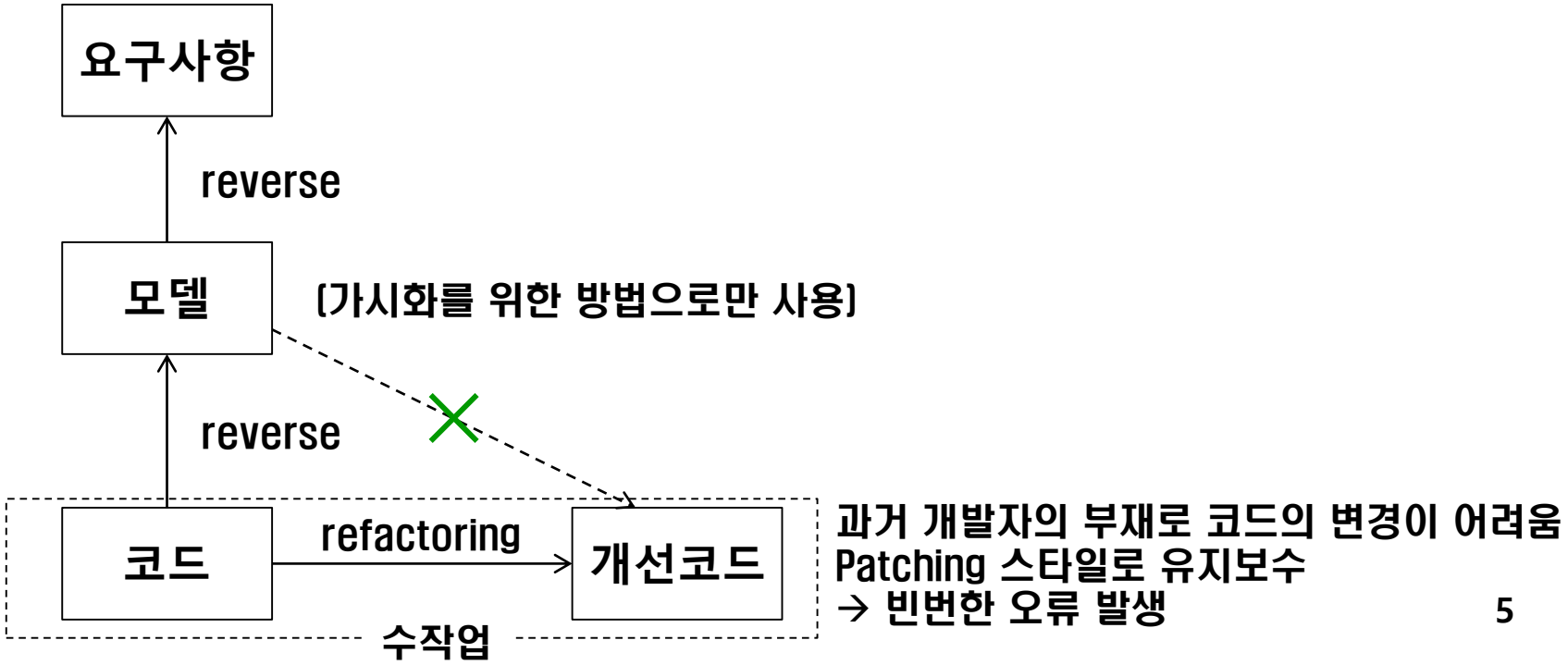
1) 새로운 시스템

- MDA기반 순공학(Forward Engineering)
  - ✓ 한번의 개발 과정으로 요구사항, 설계, 코드 등의 산출물이 발생
  - ✓ 개발 과정이 점진적으로 진행되면서, 요구사항, 설계, 코드의 변경이 되지만 산출물에 반영이 잘 되지 않음
    (즉, 설계와 코드의 동기화가 되지 않음-수작업)



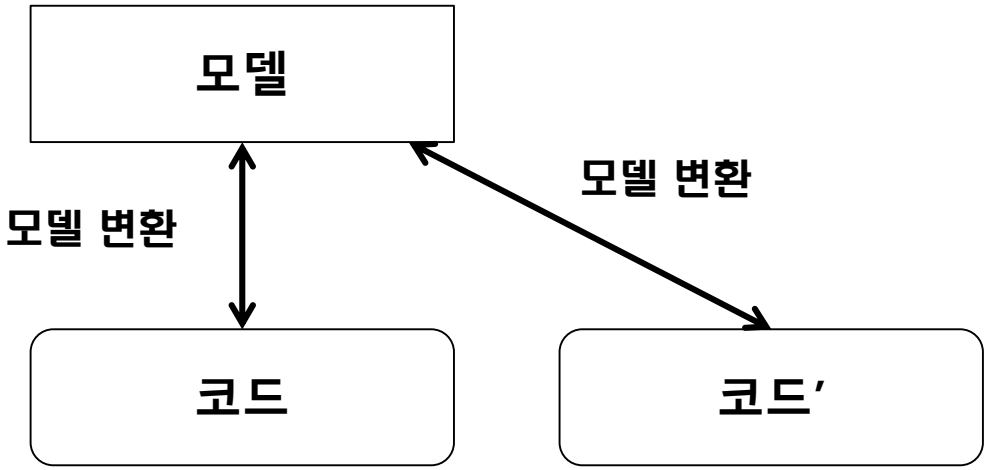
2) 기존(legacy) 시스템

- ADM기반 역공학(Reverse Engineering)
  - ✓ 레거시 코드로부터 모델을 만드는 방법
  - ✓ (일반적인) SI 업체의 현실
    - 요구사항 문서의 부재
    - 설계 도면 부재
    - 테스팅 자료 부재로 역공학을 통해 코드로부터 추출



3) 연구 목표

- 이종 모델과 코드의 동기화 및 변환
  - ✓ 이종 플랫폼 개발의 문제를 해결
  - ✓ 소프트웨어 개발 과정에서 발생하는 재사용성, 설계와 코드의 불일치, 유지보수 문제해결
- 요구사항
  - ✓ **<u>순공학에 역공학을 접목</u>**한
    1. 개발 프로세스(Process)
    2. 방법(Approach)
    3. 모델변환 언어(Language) 등이 **<u>필요함</u>**.



2. 관련연구 및 문제 정의

1) 개발 프로세스
   - ✓ MDA(Model Driven Architecture)
   - ✓ ADM(Architecture Driven Modernization)
   - ✓ 문제 정의
2) 모델 변환 방법
   - ✓ Model to Model
   - ✓ Model to Text
   - ✓ Text to Model
   - ✓ 문제 정의
3) 모델 변환 언어
   - ✓ 단방향 모델 변환 언어 : ATL, ETL, QVT-O, RubyTL
   - ✓ 양방향 모델 변환 언어 : TGG, QVT-R, JTL
   - ✓ 문제 정의

1) 개발 프로세스

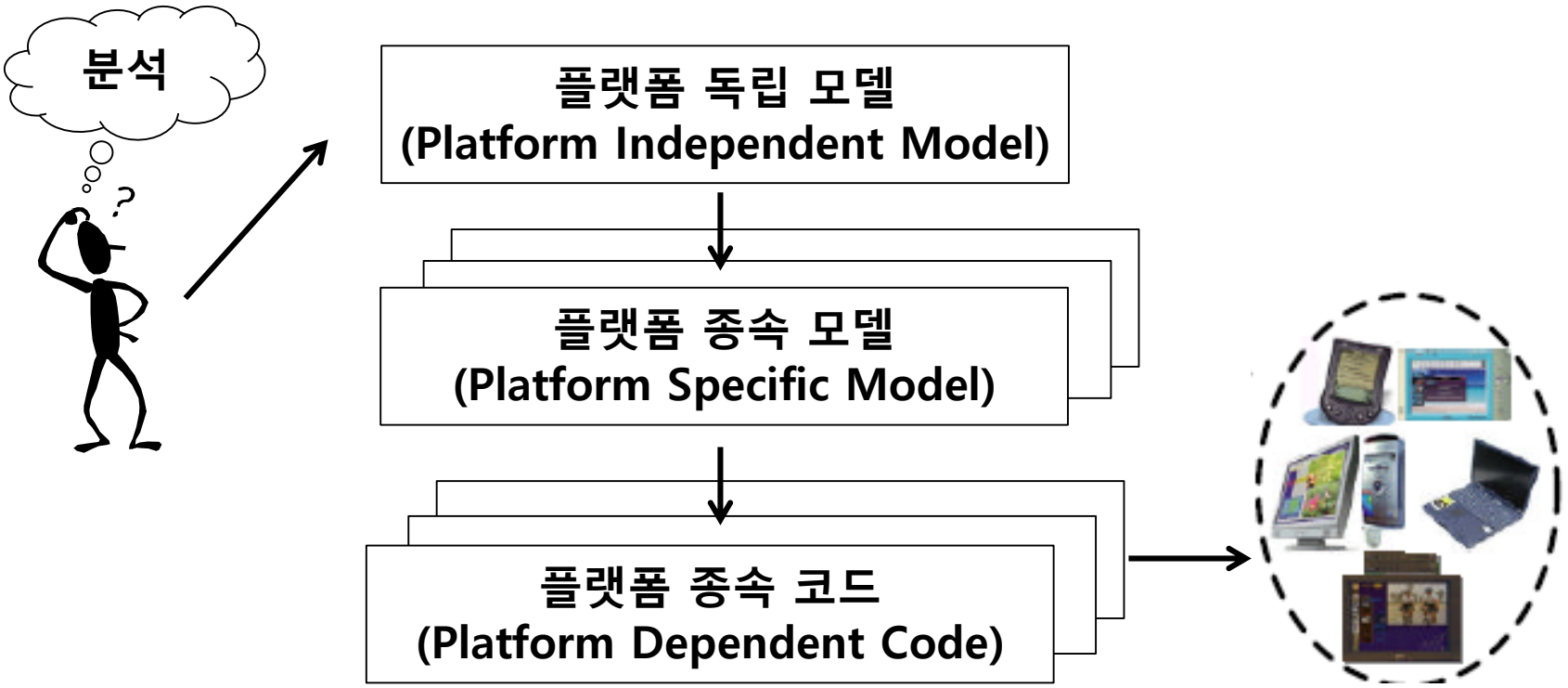
(1) MDA(Model Driven Architecture)

(2) ADM(Architecture Driven Modernization)

(3) 문제 정의

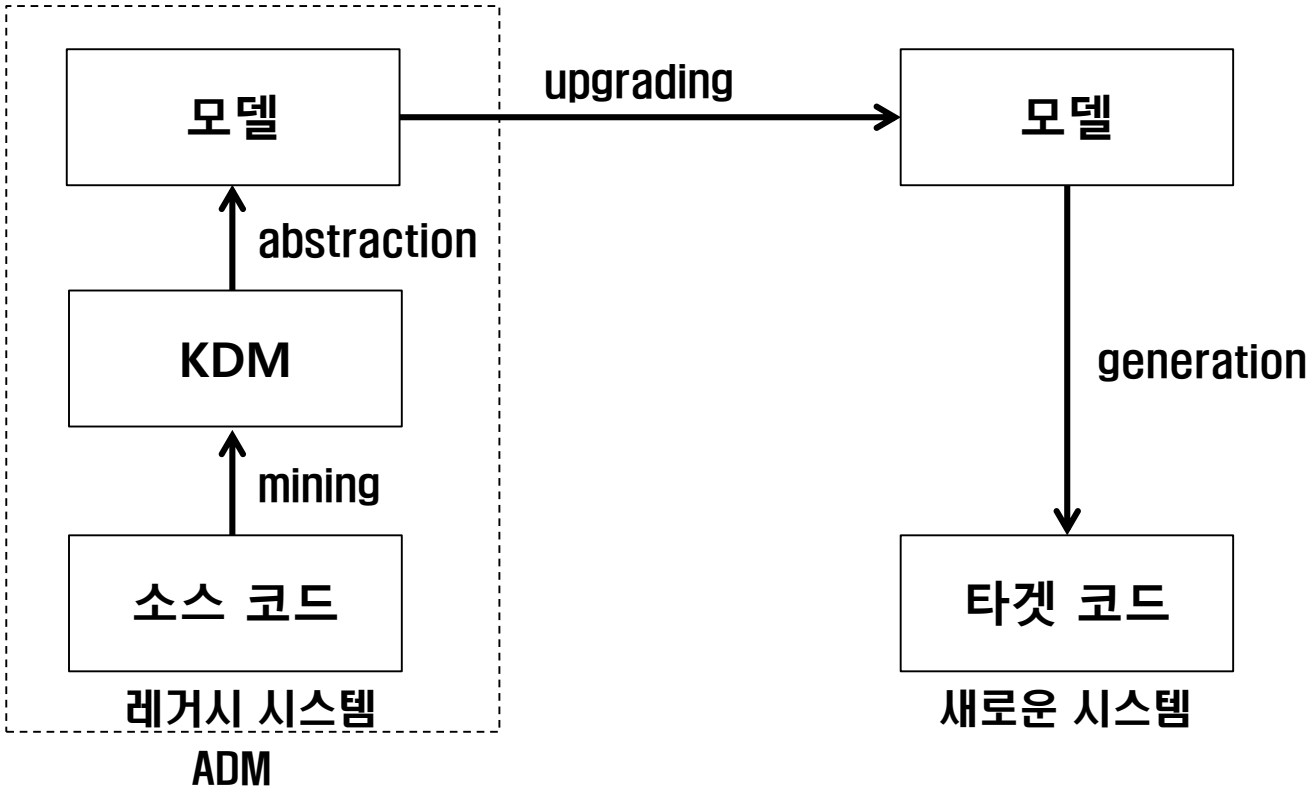
(1) MDA(Model Driven Architecture)

- MDA (순공학)
  - ✓ 플랫폼에 독립적인 설계 모델을 정의
  - ✓ 이를 기반으로 원하는 플랫폼에 맞는 소프트웨어를 생성



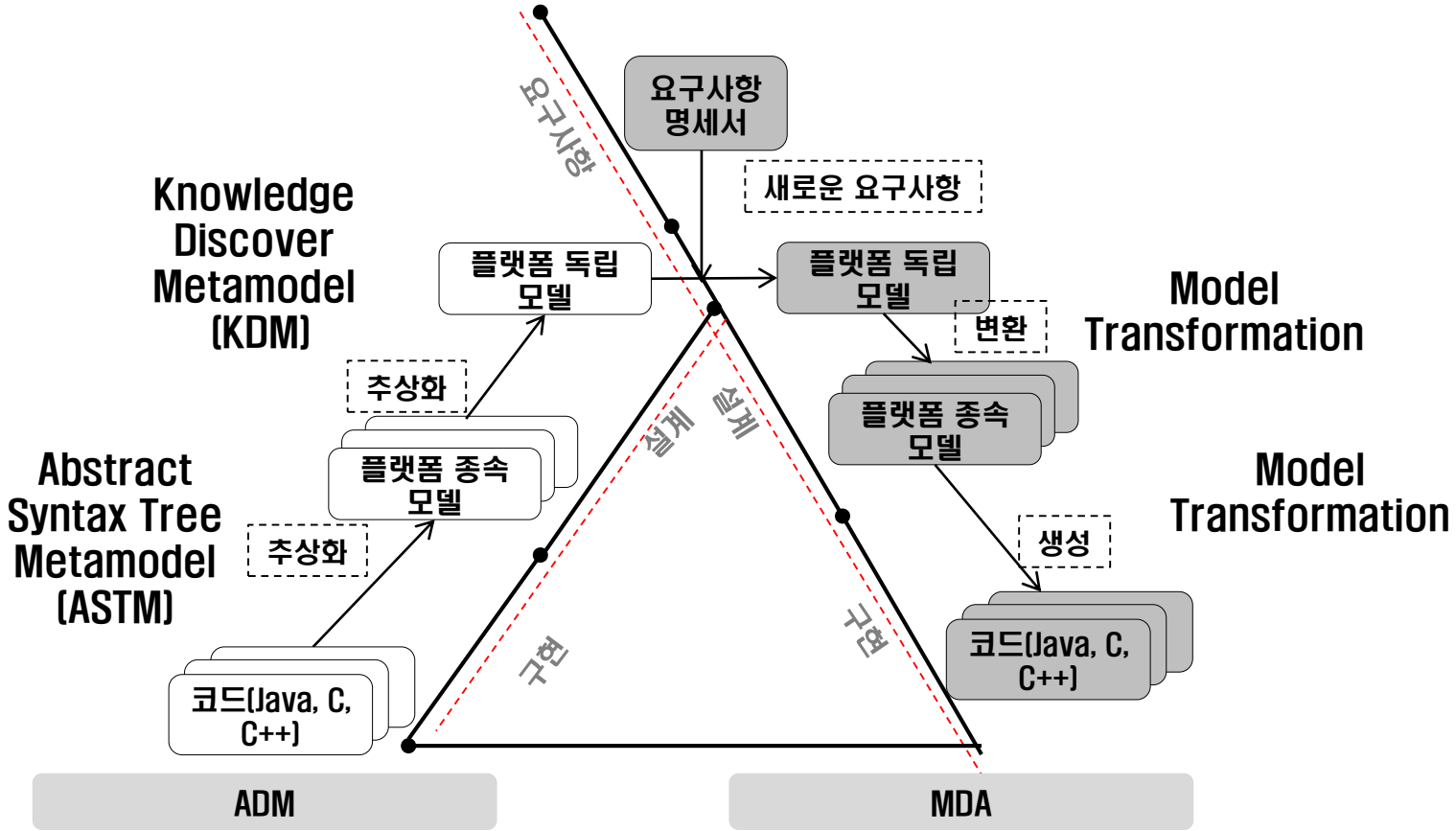
(2) ADM(Architecture Driven Modernization)

- ADM (역공학)
  - ✓ 레거시 코드를 역공학을 이용하여 모델로 변환
  - ✓ 만들어진 모델을 이용하여 새로운 시스템에 적용 가능



(3) 문제 정의

- ADM과 MDA의 접목으로,
  - ✓ (목적달성) 기존 코드를 읽어서 이종의 코드는 발생할 수 있지만
  - ✓ (비효율적) 매우 복잡한 형태의 모델변환을 사용해야 함
  - ✓ 새로운 형태의 개발 프로세스가 필요함



2) 모델 변환 방법

(1) Model to Model

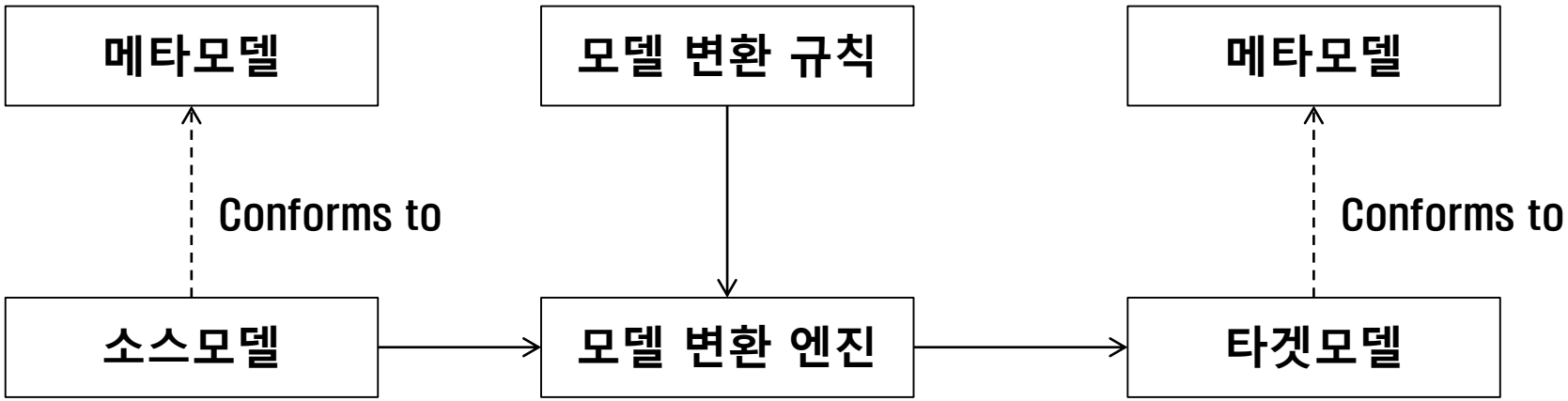
(2) Model to Text

(3) Text to Model

(4) 문제 정의

(1) Model to Model

- 메타모델을 이용하며 모델 변환 규칙을 작성하여,
  - ✓ 소스모델을 타겟모델로 변환하는 방법
  - ✓ 선언적인, 명령적인 언어가 있음
  - ✓ 대다수의 모델 변환 언어가 여기에 해당함(ATL, MTL, ETL 등)

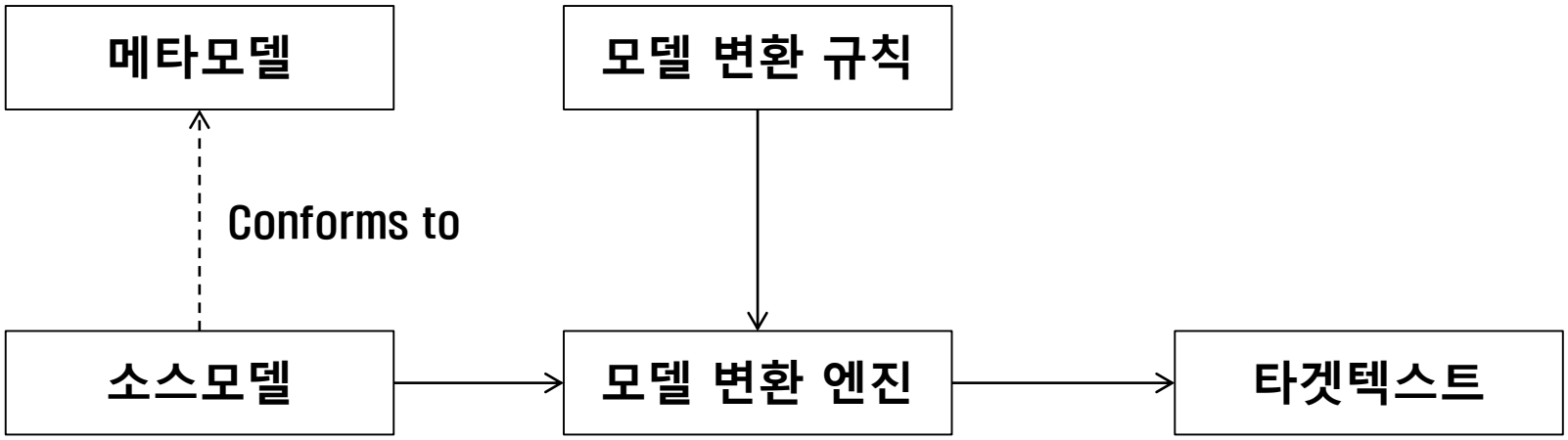


ATL을 이용한 모델 변환 규칙 작성 예

```
module Person2Contact ;
create OUT : MMb from IN : MMa ;

rule Start {
 from
   p : MMa!Person (
      p.function = 'Boss'
   )
 to
      c : MMb!Contact (
           name <- p.first_name + p.last_name
      )
}
```

(2) Model to Text

- 메타모델을 통한 모델 변환 규칙작성을 통해,
  - ✓ 소스모델을 타겟텍스트로 변환하는 방법
  - ✓ 템플릿기반 언어
  - ✓ 대표적인 언어로 Acceleo가 있음(OMG's Model-to-Text)

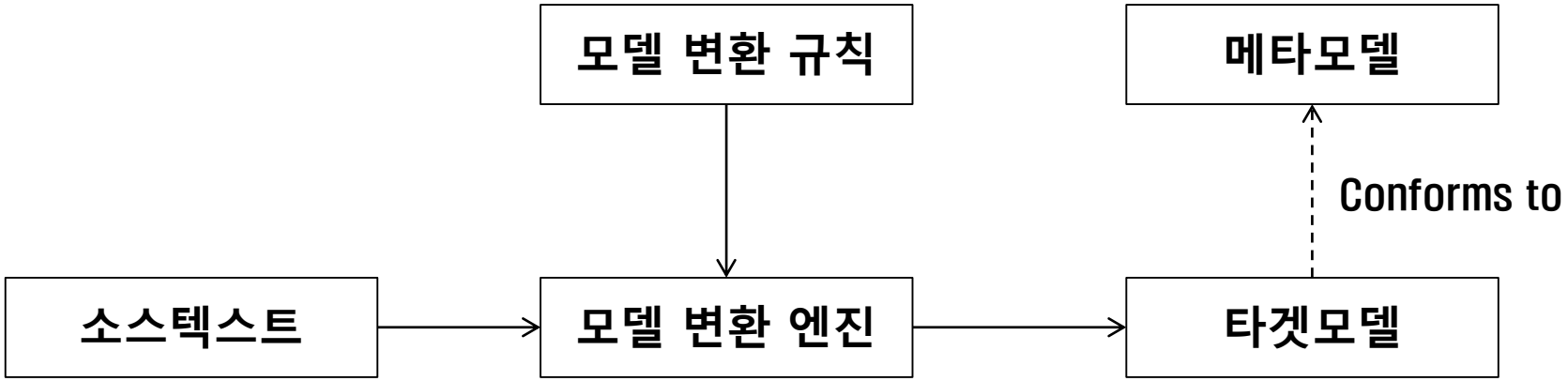


Acceleo을 이용한 모델 변환 규칙 작성 예

```
generate.mtl
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/emf/2002/Ecore')/]

[template public generate(anEClass : EClass)]

    [comment @main /]
    [file (anEClass.name + '.txt', false, 'UTF-8')]
    Hello, [anEClass.name/] world!
    [/file]

[/template]
```

(3) Text to Model

- 프로그램의 grammar와 메타모델에 대한 변환 규칙작성을 통해,
  - ✓ 소스텍스트를 타겟모델로 변환하는 방법
  - ✓ 텍스트를 읽어야 되기 때문에 파서를 이용
  - ✓ 대표적으로 Gra2MoL가 있음



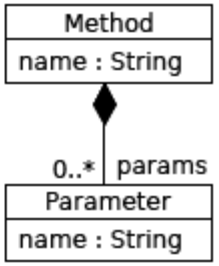
Gra2MoL을 이용한 모델 변환 규칙 작성 예

```
methodDeclaration:
  Name "(" param ")"
  ...
  ;

param:
  Type Name param?
  ;
...
```

grammar

```
rule 'example'
   from methodDeclaration mDec
   to Method
   queries
     ql : /mDec///#param;
   mappings
     name   = mDec.Name;
     params = ql;
end_rule
```

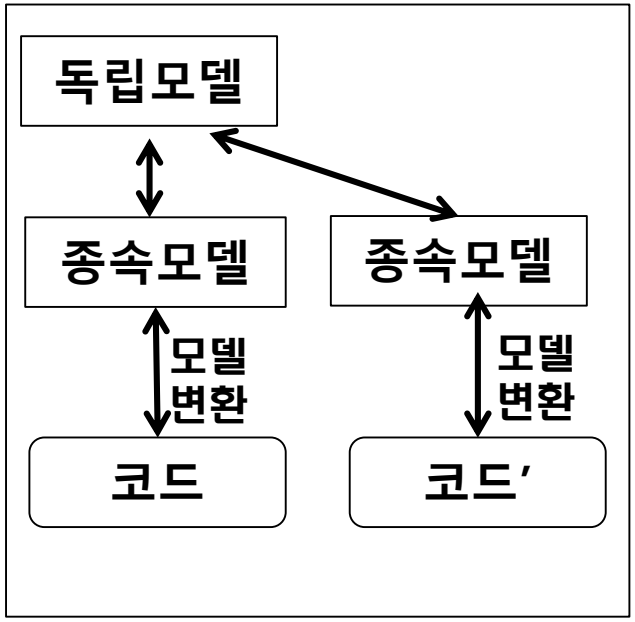
mapping rule



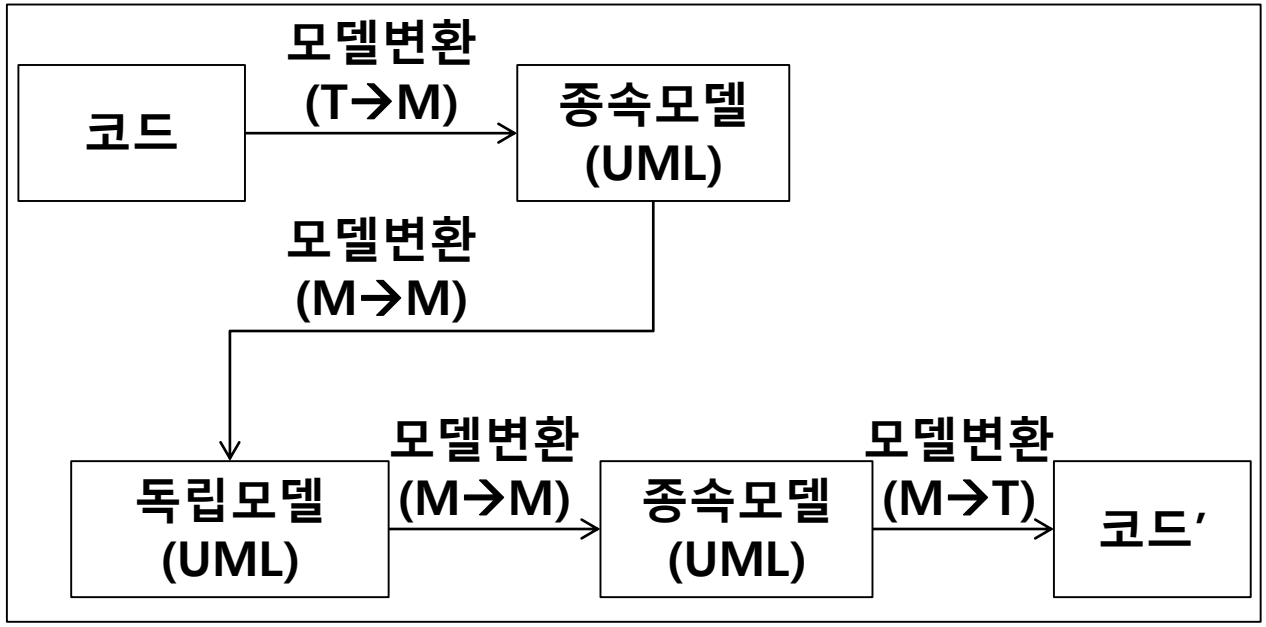
metamodel

(4) 문제 정의

- 기존 코드를 읽어서 새로운 코드로 변환 하기 위해서는
  - ✓ 6단계를 거쳐야 됨
  - ✓ 그리고 각각의 단계를 거칠 때 사용되는 언어가 다르기 때문에 각 언어를 모두 배워야 하고 구현하기가 어려움
  - ✓ M→M, M→T, T→M을 하나의 언어로 수행 가능한 방법이 필요



목표



실제 구현

3) 모델 변환 언어

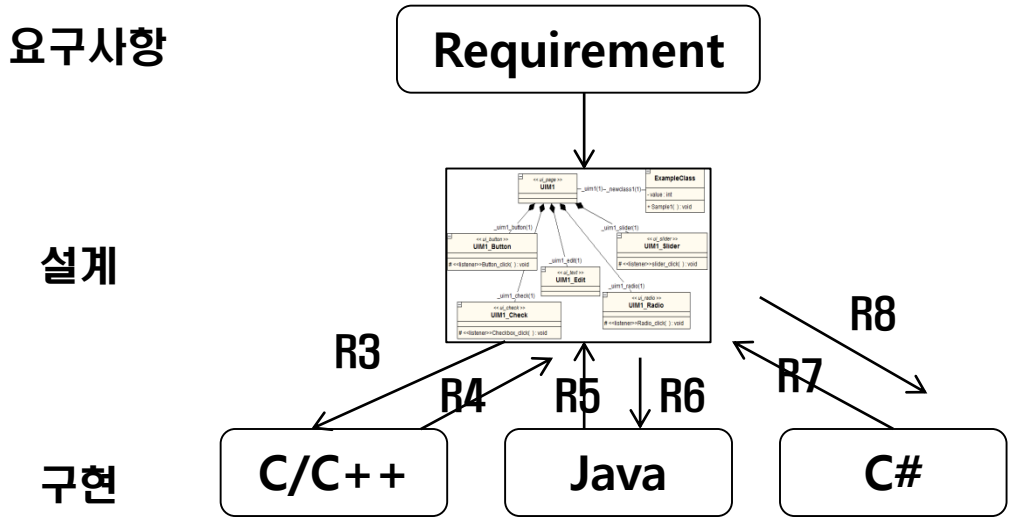
(1) 단방향 모델 변환 언어
- ✓ ATL, ETL, QVT-O

(2) 양방향 모델 변환 언어
- ✓ TGG, QVT-R, JTL

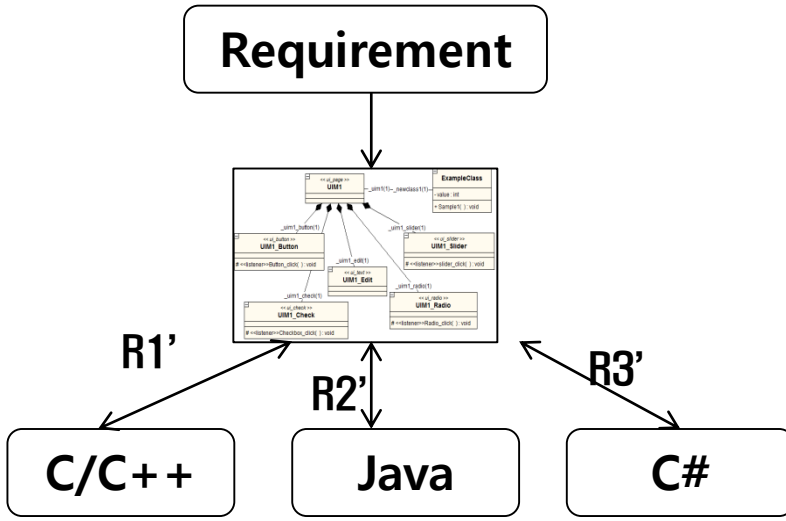
(3) 문제 정의

(1) 단방향 모델 변환 언어

- 단방향 모델 변환 언어(ATL, ETL, QVT-O)
  - ✓ 코드에서 다른 코드를 변경하기 위해서 양쪽 방향의 규칙을 따로 작성해야 됨
  - ✓ 모델이 추가될 수록 점점 더 복잡해지는 문제가 있음



단방향 모델 변환 언어



양방향 모델 변환 언어

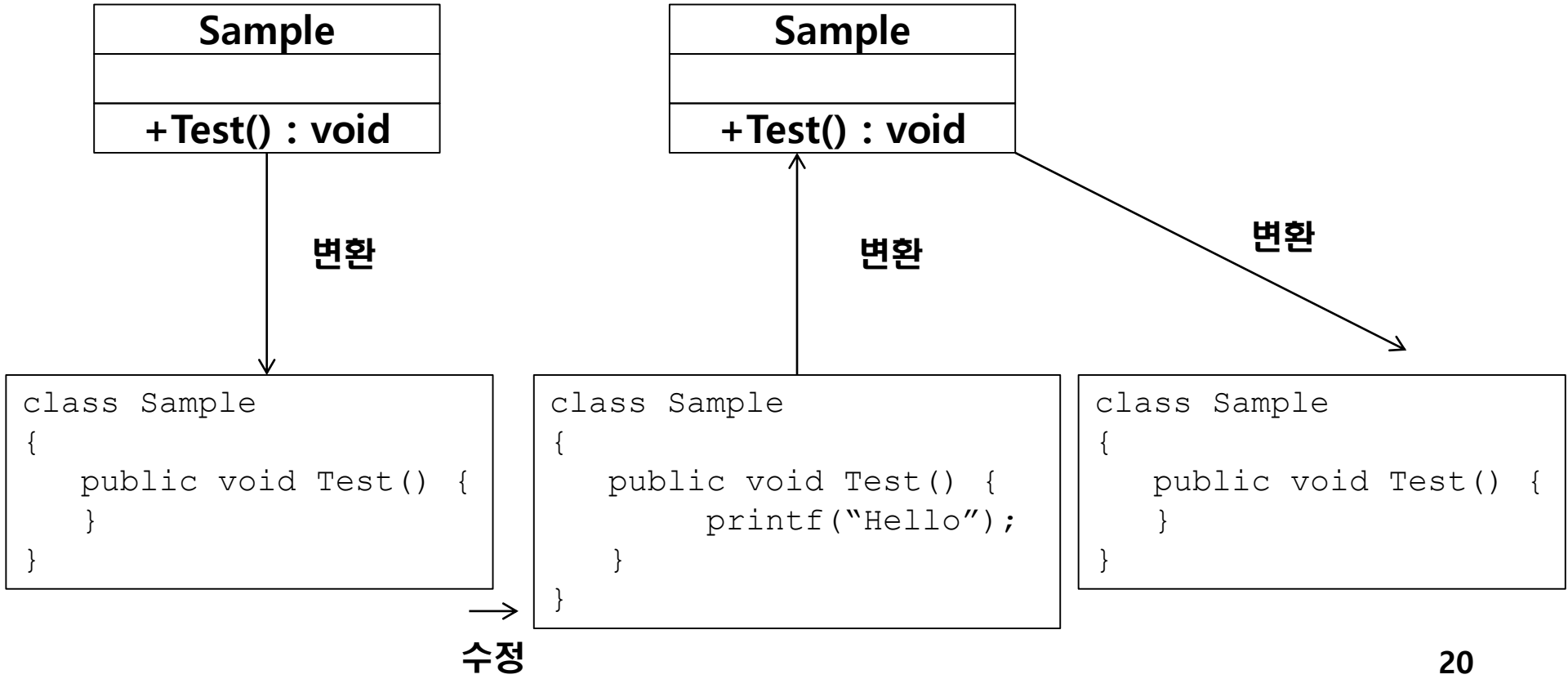
(2) 양방향 모델 변환 언어

- TGG(Triple Grape Grammars)
  - ✓ 그래프를 이용하여 양방향 모델 변환이 가능
  - ✓ 양모델의 추상화 수준이 다르면 정보 손실 될 수 있음
- QVT-R(Query View Transformation Relations)
  - ✓ 양모델의 추상화 수준이 다르면 정보 손실 될 수 있음
- JTL(Janus Transformation Language)
  - ✓ 추론 규칙을 적용해 추상화 수준이 다를 때 근사치로 변환 가능
  - ✓ 정확하지 않고 모델 변환 결과가 여러 개임

- 이종 코드로 변환을 위해서는 정보의 손실이 없는 새로운 모델 변환 언어가 필요

(3) 문제 정의

- 정보 손실의 문제
  - ✓ 동등한 수준의 모델은 기존 양방향 언어를 사용해도 문제 없음
  - ✓ 그러나 추상화 수준이 다른 모델을 사용하면 변환시 정보 손실



기존 모델 변환 언어들의 비교

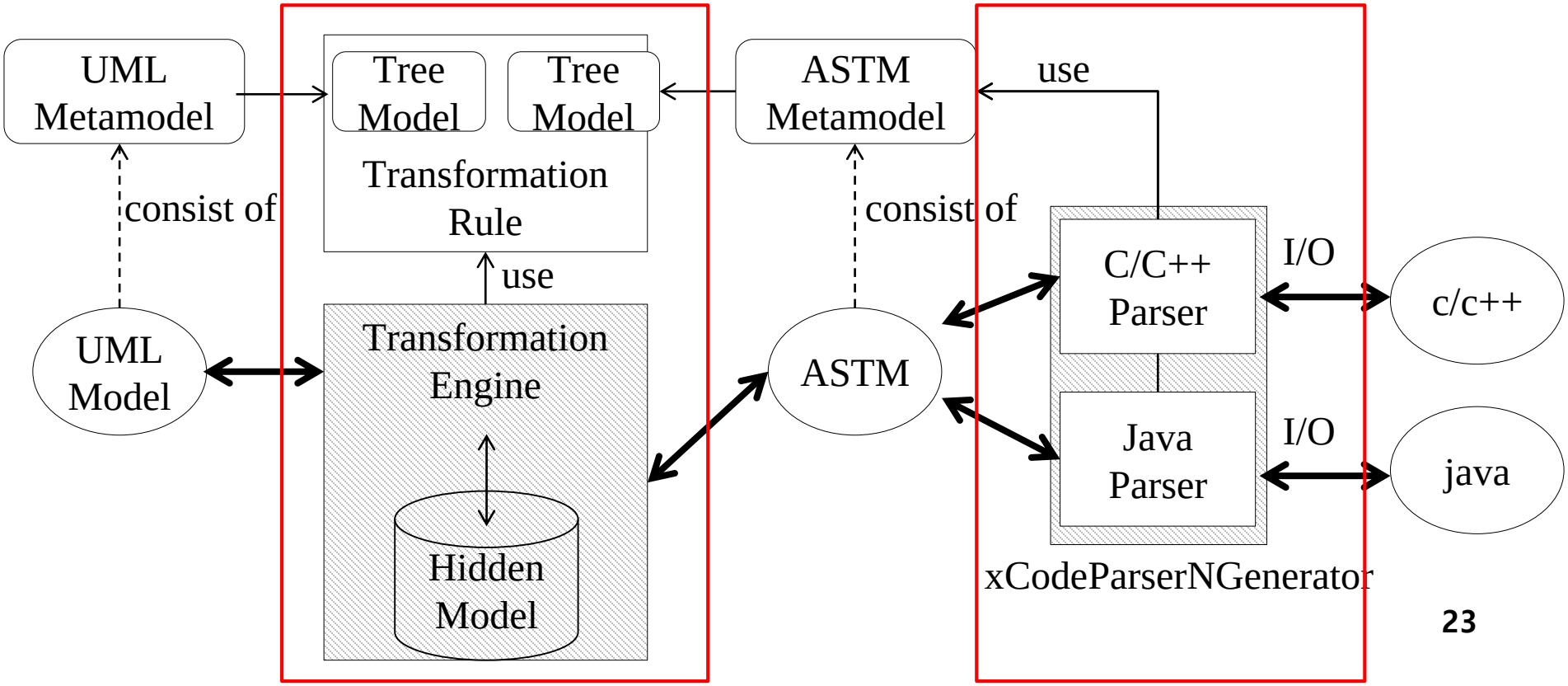
Direction	Language / Framework	Language Coverage									Type		
		Input Element	Output Element	Negative Condition	Positive Condition	Assignment	Rule Inheritance	Declarative Parts	Imperative Pars	OCL Support	Model to Model	Model to Text	Text to Model
Unidirectional	ATL	1	1..*	Y	Y	Y	Y	Y	Y	Y	Y	N	N
	ETL	1	1..*	Y	Y	Y	Y	Y	Y	Y	Y	N	N
	QVT-O	1	1..*	Y	Y	Y	Y	N	Y	Y	Y	N	N
Bidirectional	TGG	1..*	1..*	Y	Y	Y	N	Y	N	P	Y	N	N
	QVT-R	1..*	1..*	Y	Y	Y	Y	Y	Y	Y	Y	N	N
	JTL	1..*	1..*	Y	Y	Y	Y	Y	Y	Y	Y	N	N
	BMTL	1..*	1..*	Y	Y	Y	N	Y	Y	N	Y	Y	Y

3. 제안한 양방향 모델 변환 프레임워크

1) 제안한 양방향 모델 변환 방법
   - ✓ 제안한 양뱡항 모델 변환 언어
   - ✓ 제안한 양방향 모델 변환 엔진
2) 제안한 코드 생성 및 리버스 방법
   - ✓ Abstract Syntax Tree Metamodel(ASTM)
   - ✓ 제안한 코드 리버스 방법
   - ✓ 제안한 코드 생성 방법

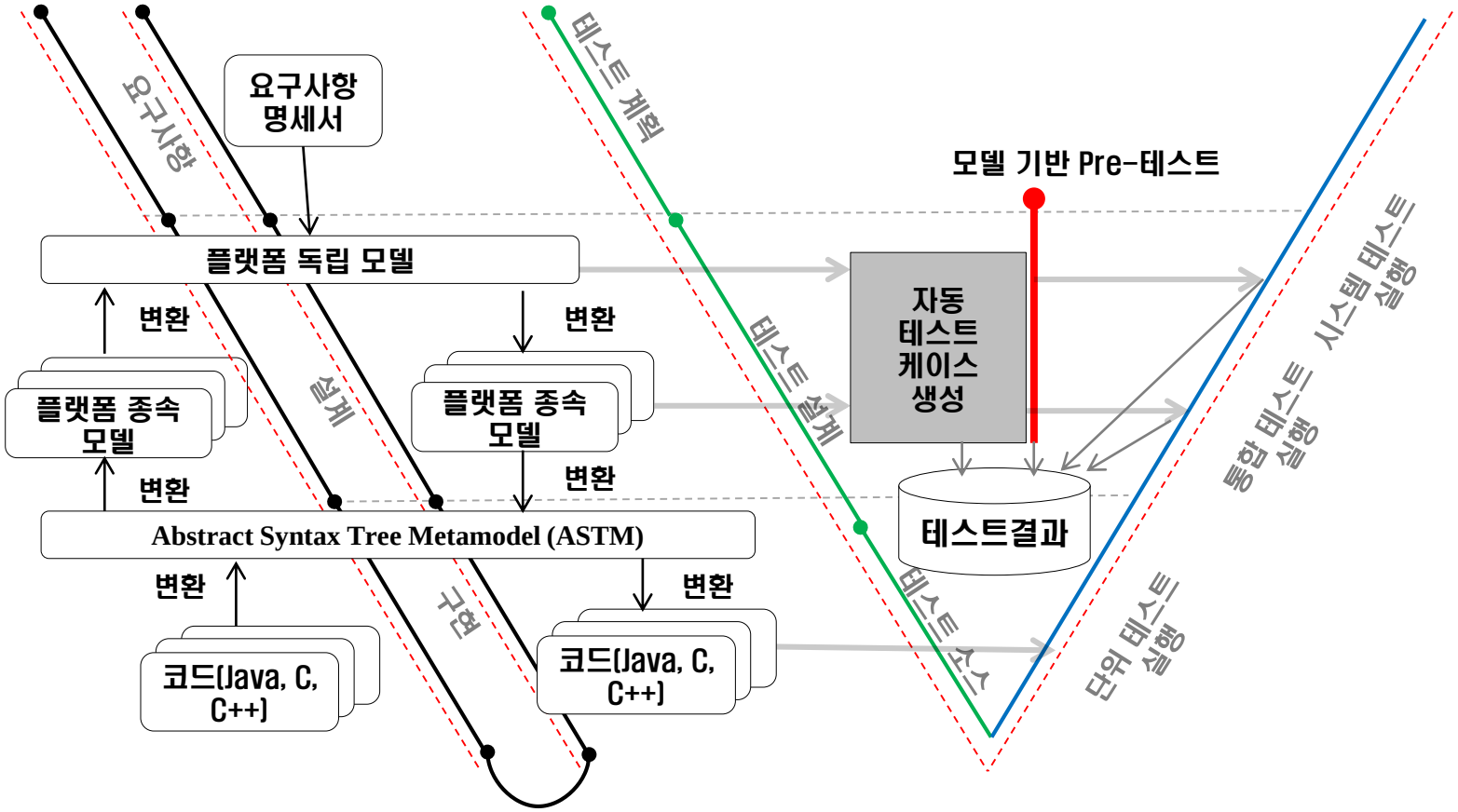
3. 제안한 양방향 모델 변환 프레임워크

- 제안한 프레임워크
  - ✓ (프로세스) MDA와 ADM을 병합한 프로세스
  - ✓ (방법) M→M, M→T, T→M을 하나의 언어에서 가능
  - ✓ (언어) 양방향 모델 변환 언어 : ASTM과 Hidden Model을 사용하여 정보손실 방지



3. 제안한 양방향 모델 변환 프레임워크

➢ 제안한 개발 프로세스(MDA와 ADM을 병합)

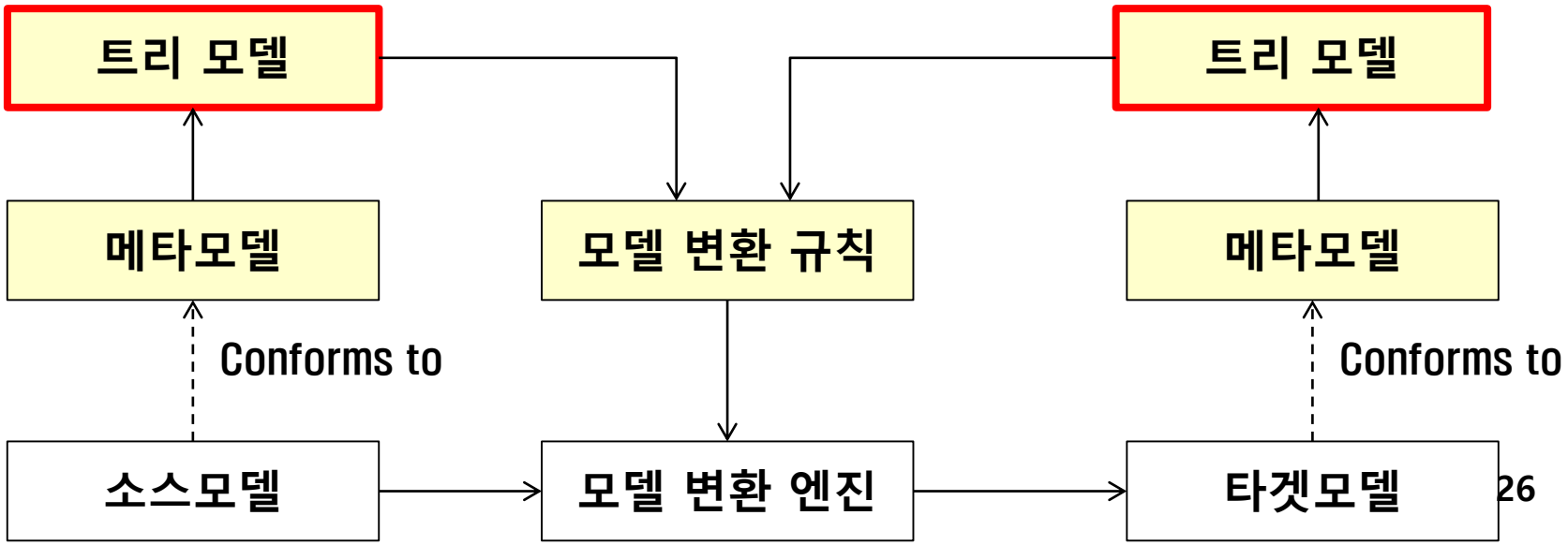


1) 제안한 양방향 모델 변환 방법

(1) 제안한 양방향 모델 변환 언어
(2) 제안한 양방향 모델 변환 엔진

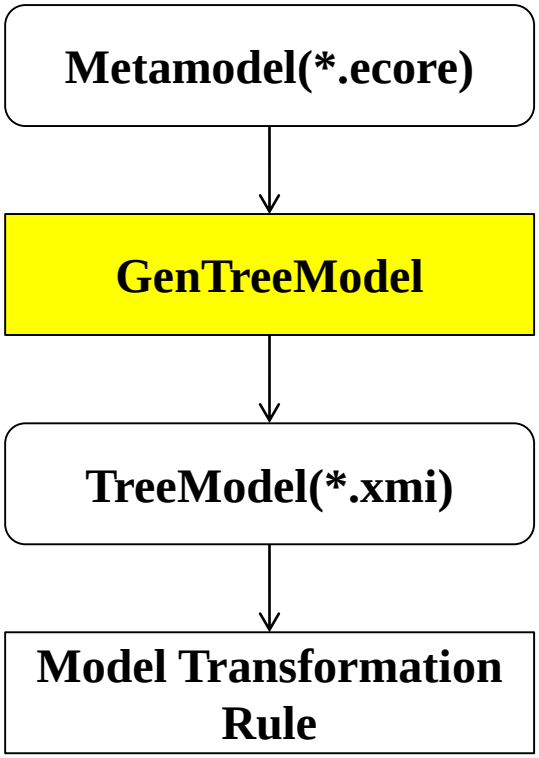
(1) 제안한 양방향 모델 변환 언어

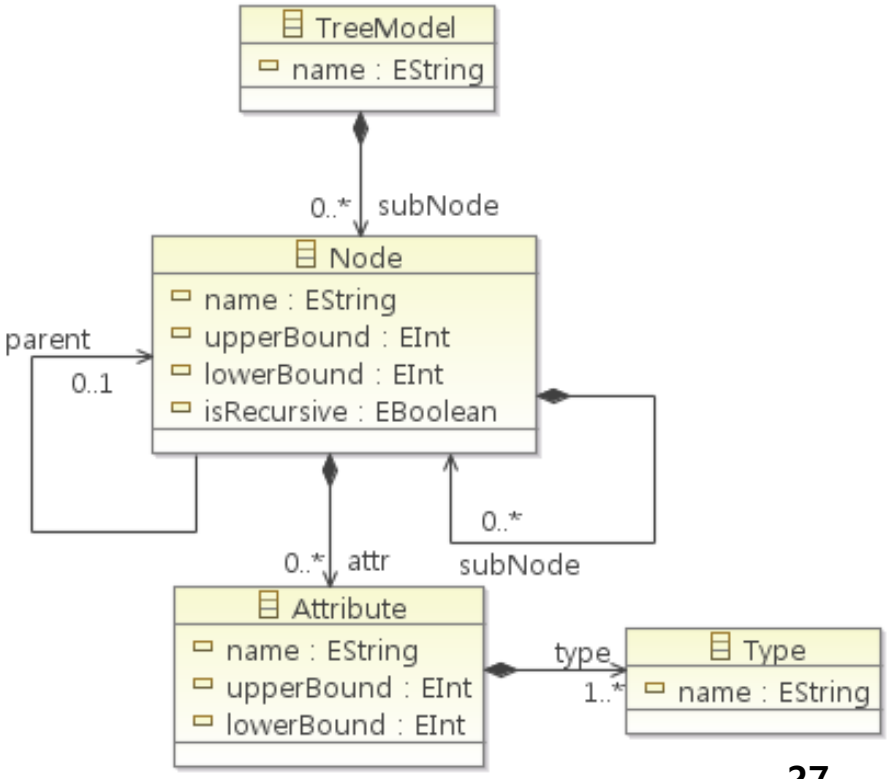
- 트리기반의 양방향 모델 변환 언어
  - ✓ 메타모델은 XML을 기반으로 트리 형태의 데이터 구조를 가짐
  - ✓ 정보 손실을 방지 하기 위해서는 모든 메타클래스의 변환이 필요함 (모든 메타모델의 요소(추상 클래스)의 체크는 필요 없음)
  - ✓ 양방향을 위해 원자(Atomic) 단위의 데이터 변환 후 이를 재 배열하는 방식을 사용
  - ✓ 트리 기반의 언어를 제안
- 변환 과정



(1) 제안한 양방향 모델 변환 언어

- 메타모델의 복잡성
  - ✓ 상속과 연관 관계가 선으로 되어 있어 알아보기 어려움
  - ✓ 양방향에서 모든 모델의 요소가 매치되어야 하기 때문에 메타모델의 구조를 쉽게 파악할 수 있어야 됨





TreeModel Metamodel

(1) 제안한 양방향 모델 변환 언어

➢ Tree Model의 노드(EBNF)

<NODE> := [“[“ <MULTIPLE> “]”] [<NAME>] “:” <TYPE> “(“<VARIABLES> “)”

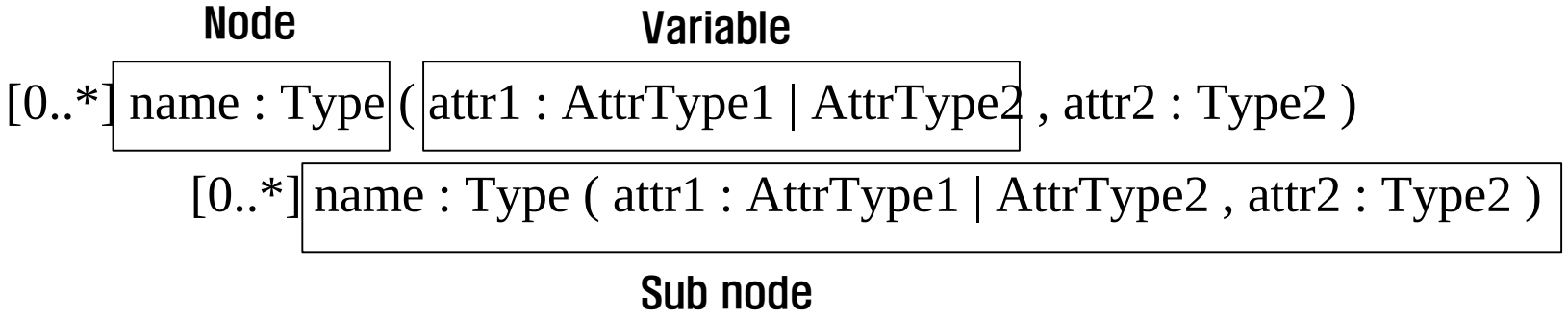
<MULTIPLE> := “0..1” | “0..*” | “1..*”

<VARIABLES> := <VARIABLE> (“,” <VARIABLE>)*

<VARIABLE> := <NAME> “:” <TYPES>

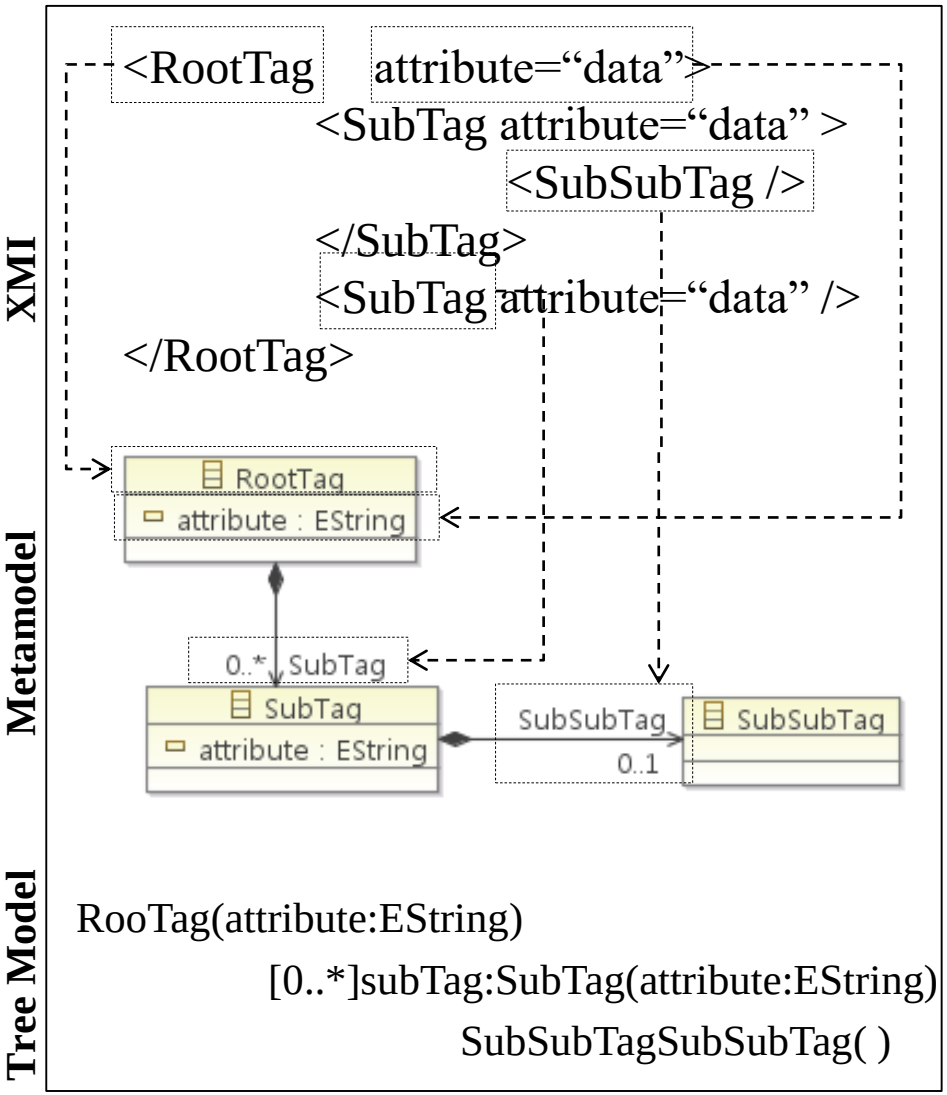
<TYPES> := <TYPE> (“|” <TYPE>)*

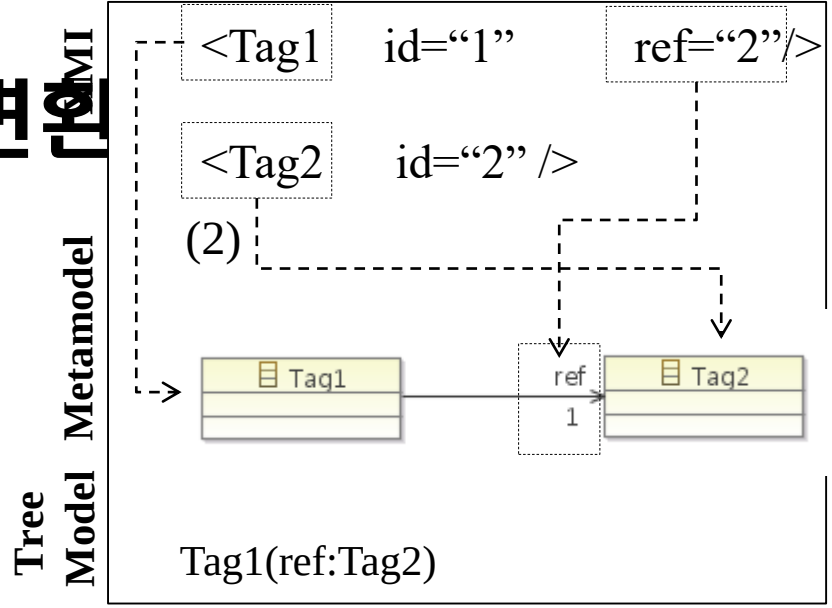
Example)

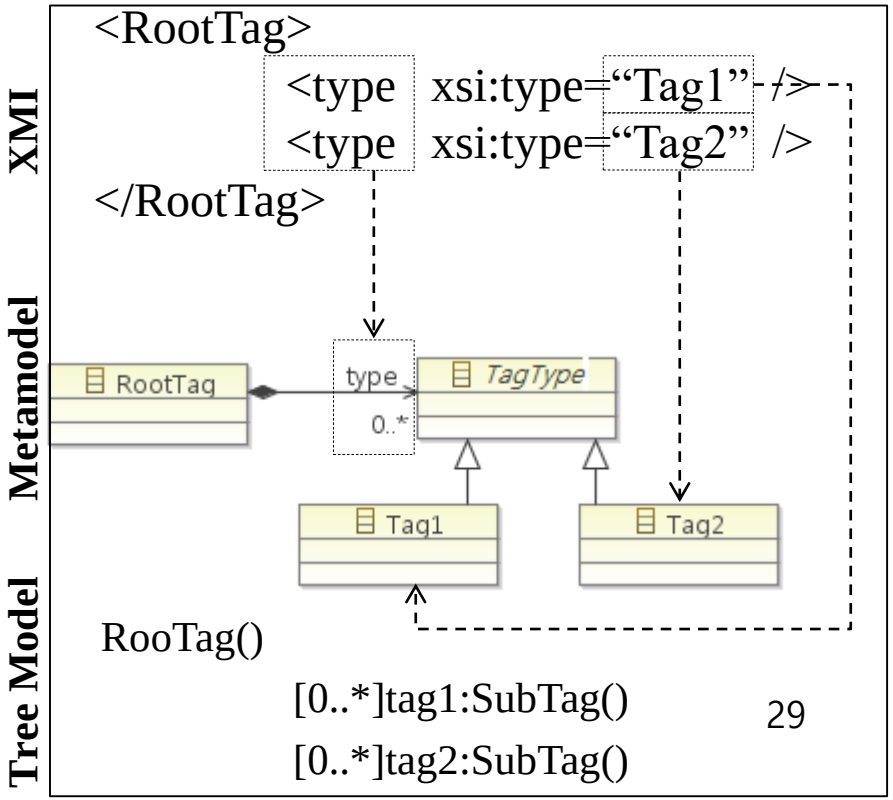


(1) 제안한 양방향 모델 변환

- Tree Model 변환 방법





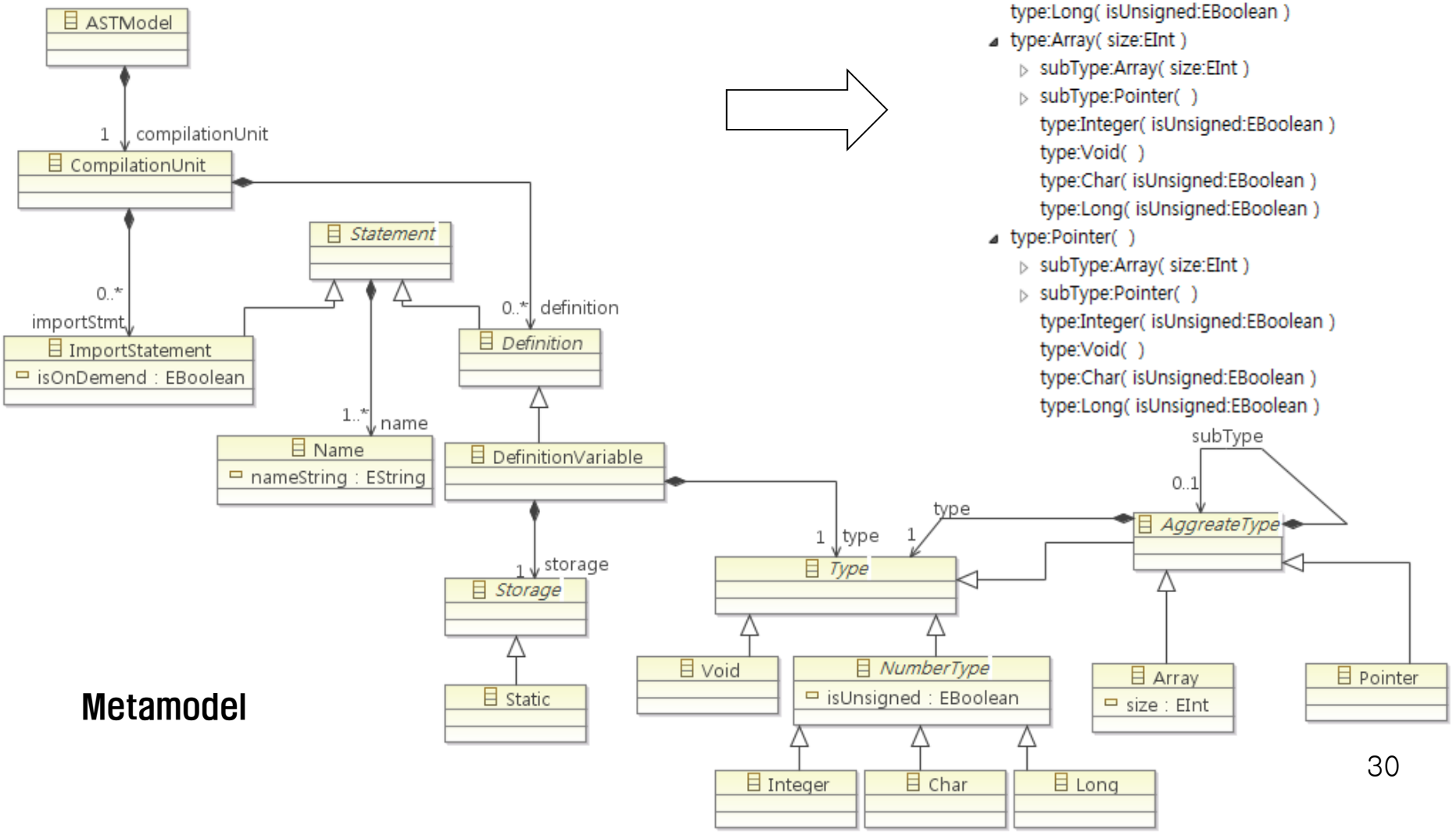


(1) 제안한 양방향 모델 변환

- Example

Tree Model

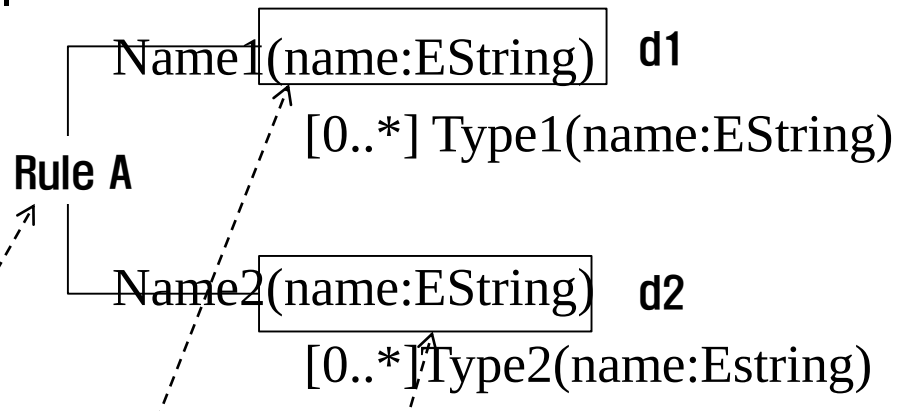
- :ASTModel( )
  - compilationUnit:CompilationUnit( )
    - [0..*]importStmt:ImportStatement(isOnDemend:EBoolean)
      - [1..*]name:Name(nameString:EString)
    - [0..*]definition:DefinitionVariable( )
      - [1..*]name:Name(nameString:EString)
      - storage:Static( )
      - type:Integer(isUnsigned:EBoolean)
      - type:Void( )
      - type:Char(isUnsigned:EBoolean)
      - type:Long(isUnsigned:EBoolean)
      - type:Array(size:EInt)
        - subType:Array(size:EInt)
        - subType:Pointer( )
        - type:Integer(isUnsigned:EBoolean)
        - type:Void( )
        - type:Char(isUnsigned:EBoolean)
        - type:Long(isUnsigned:EBoolean)
      - type:Pointer( )
        - subType:Array(size:EInt)
        - subType:Pointer( )
        - type:Integer(isUnsigned:EBoolean)
        - type:Void( )
        - type:Char(isUnsigned:EBoolean)
        - type:Long(isUnsigned:EBoolean)



Metamodel

(1) 제안한 양방향 모델 변환 언어

- Graphical



- Textual

```
domain m1:"metamodel1.ecore";
domain m2:"metamodel2.ecore";
rule A{
d1: m1!Name1.name
d2: m2!Name2.name
    mapping {
        d1 <--> d2
    }
}
```

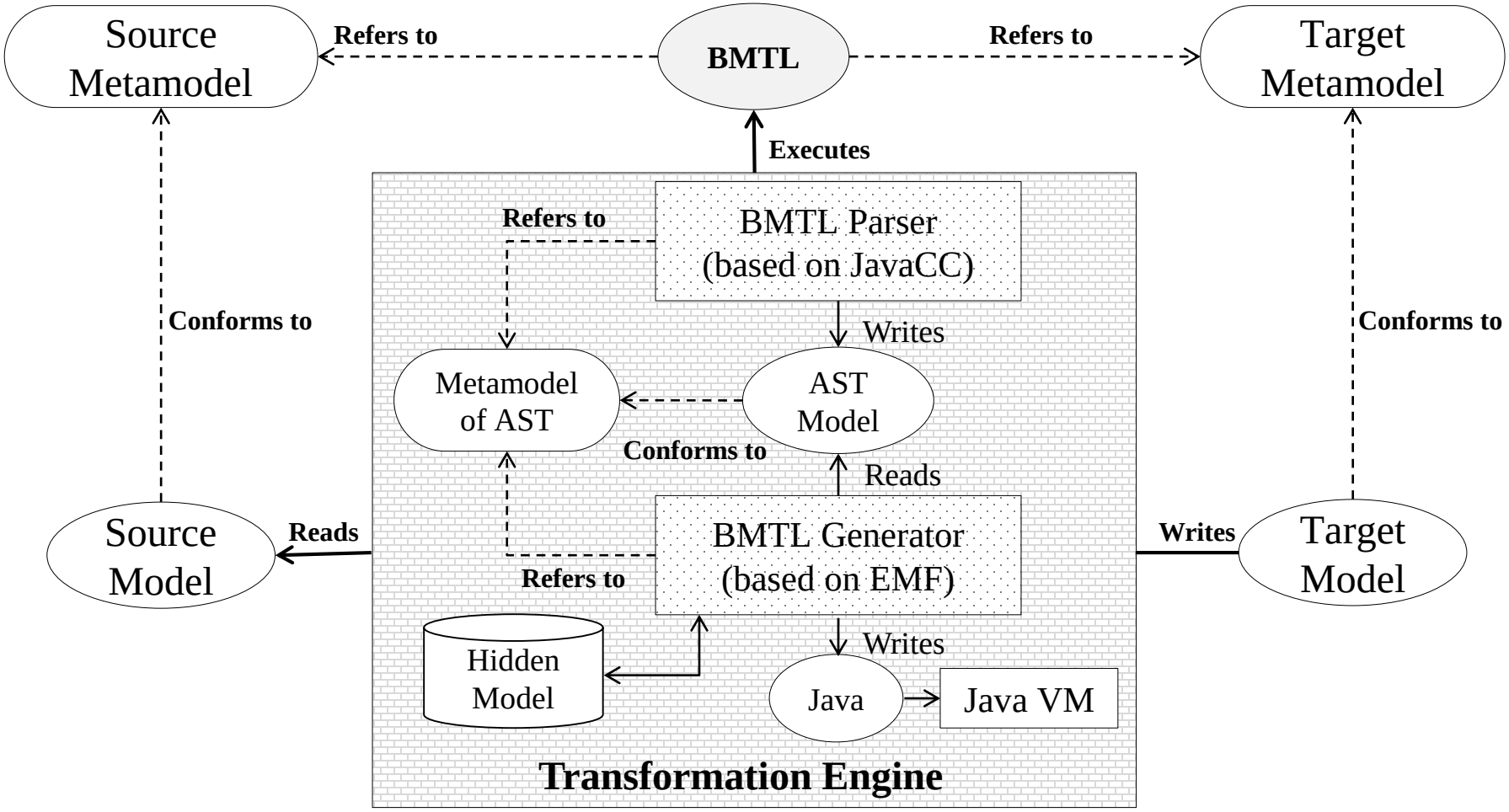
(1) 제안한 양방향 모델 변환 언어

- Language Format

```
[ domain <DOMAIN_NAME>:"<FILE_NAME>"; ]+

[ rule <RULE_NAME> {

[ <NAME> : <DOMAIN_NAME>!<CLASS_NAME>.<ATTRIBUTE_NAME> ]*

    mapping {
        [ <NAME> <-[<CONDITION>]-> <NAME> [:: <ACTION_NAME>] ]*
    }

    [ do(<ACTION_NAME>) {@
        [ <CODE> ]*
    @} ]*
} ]+
```

(2) 제안한 양방향 모델 변환 엔진

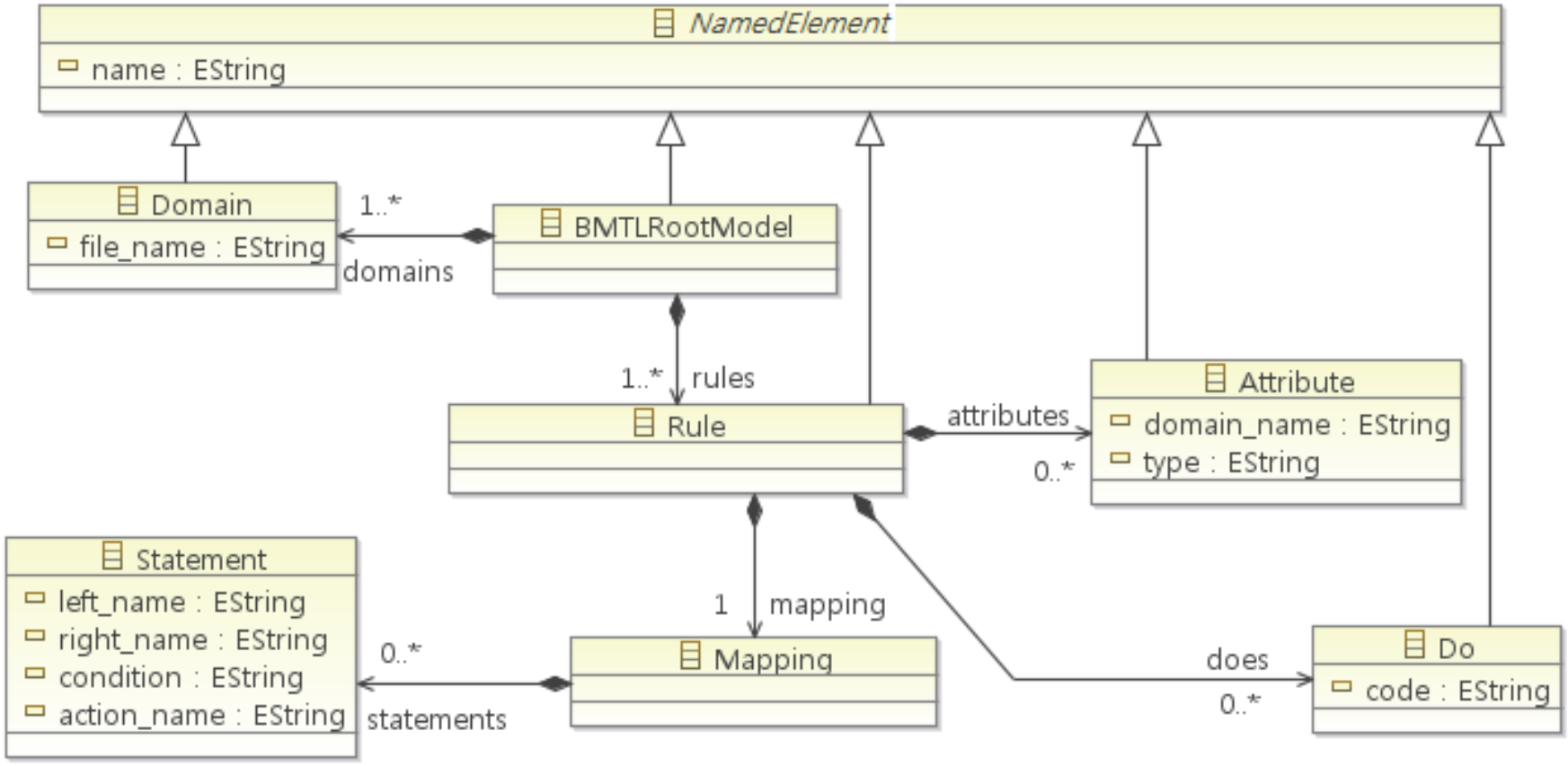
➢ 모델 변환 엔진의 구조



BMTL: Bidirectional Model Transformation Language, EMF: Eclipse Modeling Framework, AST: Abstract Syntax Tree

(2) 제안한 양방향 모델 변환 엔진

- BMTL AST Metamodel



(2) 제안한 양방향 모델 변환 엔진

- Bidirectional Model Transformation Language (BMTL)의 EBNF

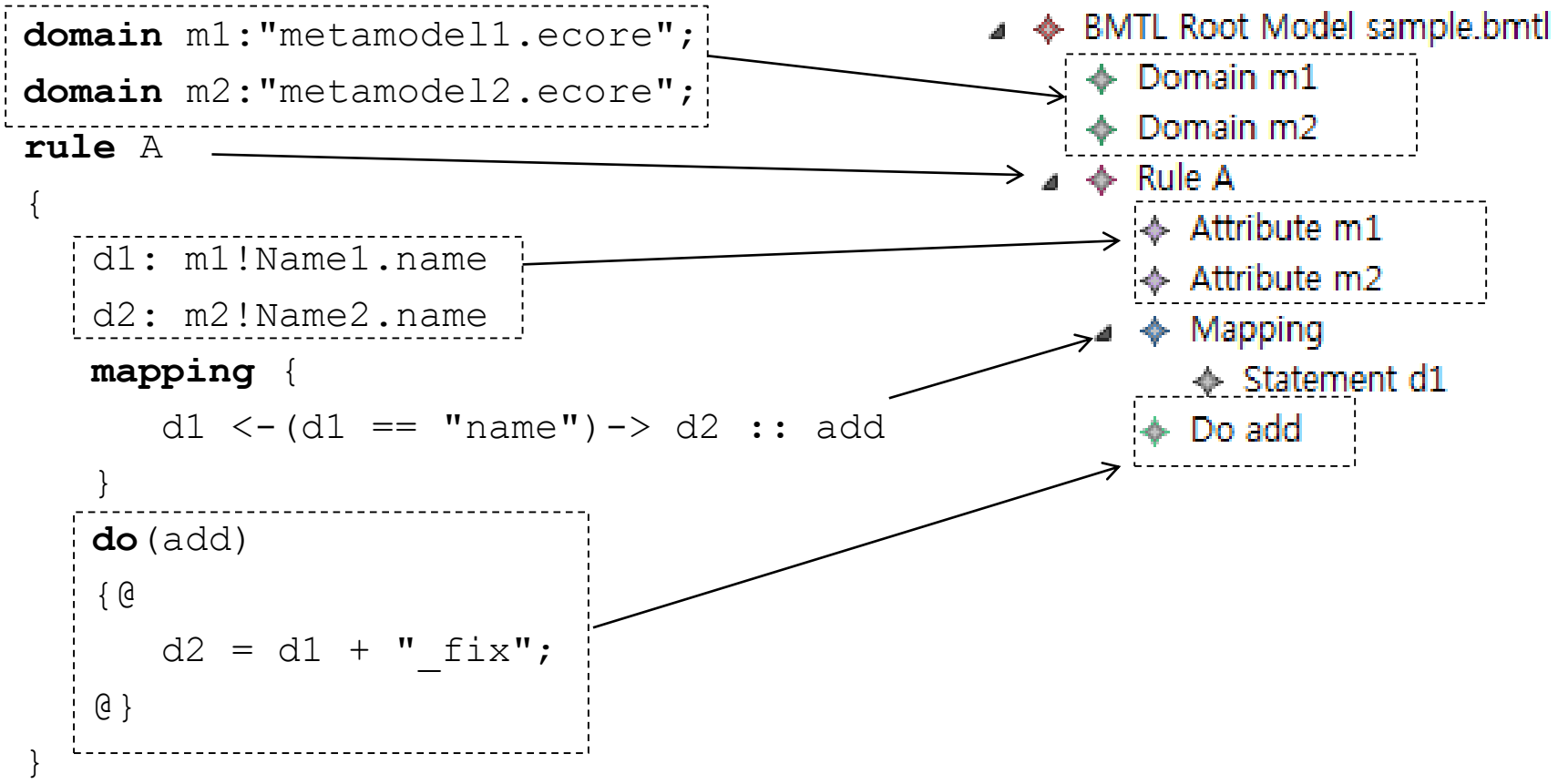
```
<BMTL> ::= <domains> <rules>
<domains> ::= ("domain" <domain>)+
<domain> ::= <name> ":" <LITERAL_STRING> ";"
<rules> ::= "rule" <name> "{" <attributes> <mapping> <does> "} "
<attributes> ::= ( <attribute> )*
<attribute> ::= <name> ":" <name> ":" <type>
<type> ::= <name> ("." <name> )*
<mapping> ::= "mapping" "{" <statements> "} "
<statements> ::= ( <statement> )*
<statement> ::= <name> "<-" ["(" <expr> ")"] "->" <name> [":" <name>]
<does> ::= ( <do_stmt> )*
<do_stmt()> ::= "do" "(" <name> ")" "{@" <CODE_STRING> "@}"
<name> ::= <IDENTIFIER>
```

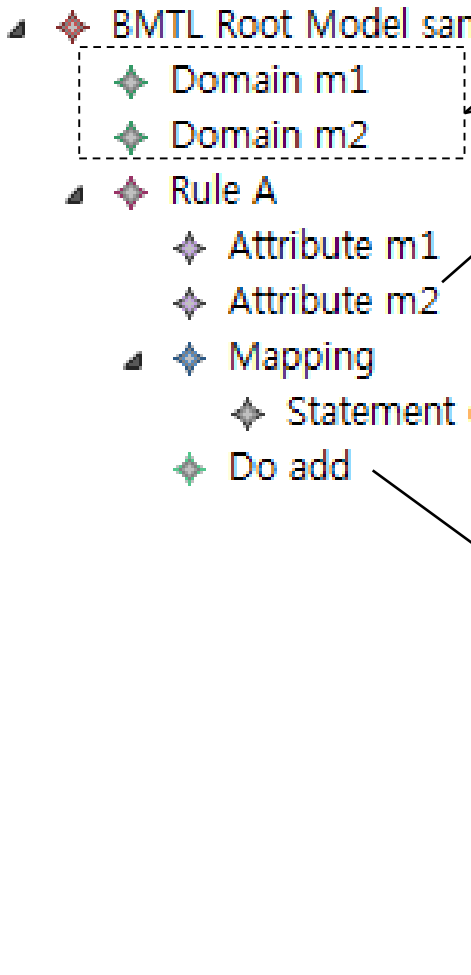
(2) 제안한 양방향 모델 변환 엔진–실행 코드 생성

```
[ domain <DOMAIN_NAME>:"<FILE_NAME>" ]+
[ rule <RULE_NAME> {
[ <NAME> : <DOMAIN_NAME>!<CLASS_NAME>.<ATTRIBUTE_NAME> ]*
    mapping {
        [ <NAME> <-[<CONDITION>]-> <NAME> [:: <ACTION_NAME>] ]*
    }
    [ do(<ACTION_NAME>) {@
        [ <CODE> ]*
    @} ]*
} ]+
public class <RULE_NAME> {
   private boolean direction;
   private Resource <DOMAIN_NAME>
   private <CLASS_NAME>.<ATTRIBUTE_NAME> <NAME>;
   public <RULE_NAME>(boolean dir) {
      <DOMAIN_NAME> = emf_load(<FILE_NAME>); btml_mappiing();
      direction = dir;
   }
   private btml_mapping() {
      if(direction) {
         if(<CONDITION>){<NAME> = <NAME>; btml_<ACTION_NAME>();}
      }
      else{
         if(<CONDITION>){<NAME> = <NAME>; btml_rev_<ACTION_NAME>();}
      }
   }
   private void btml_<ACTION_NAME> () { <CODE> }
   private void btml_rev_<ACTION_NAME> () { <CODE> }
}
```

(2) 제안한 양방향 모델 변환 엔진

➢ BMTL와 메타클래스와의 관계

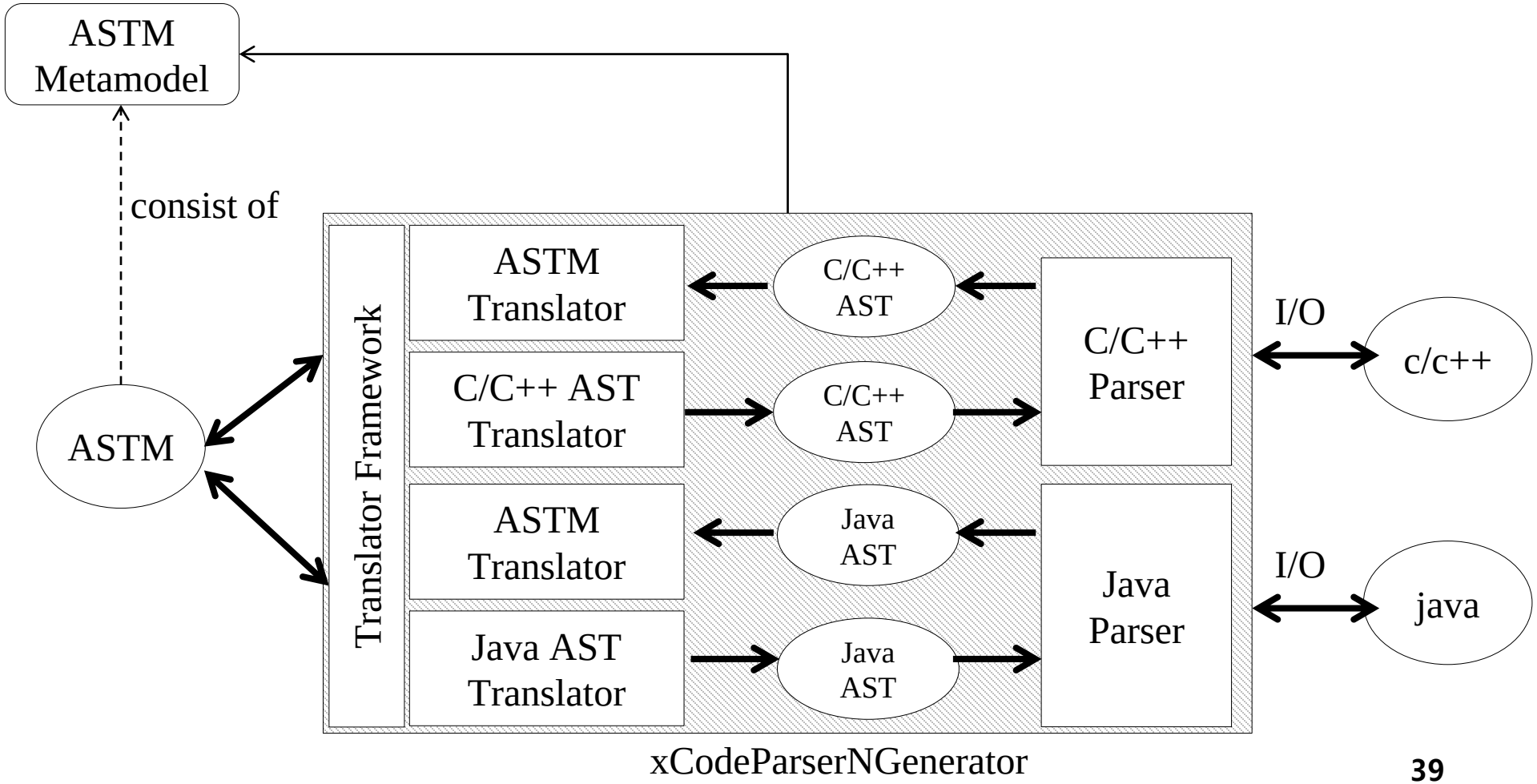





```
public class A {
    private boolean direction;
    private Resource m1;
    private Resource m2;
    private Name1.name m1;
    private Name2.name m2;
    public void A(boolean dir) {
        m1 = emf_load("metamodel1.ecore");
        m2 = emf_load("metamodel2.ecore");
        btml_mapping();
        direction = dir;
    }
    private void btml_mapping() {
        if (direction) {
            if (d1 == "name") {
                d1 = d2;
                btml_add();
            }
        } else {
            if (d1 == "name") {
                d2 = d1;
                btml_rev_add();
            }
        }
    }
    private void btml_add() {
        d2 = d1 + "_fix";
    }
    private void btml_rev_add() {
        d2 = d1 + "_fix";
    }
}
```

2) 제안한 코드 생성 및 리버스 방법

- xCodeParserNGenerator



2) 제안한 코드 생성 및 리버스 방법

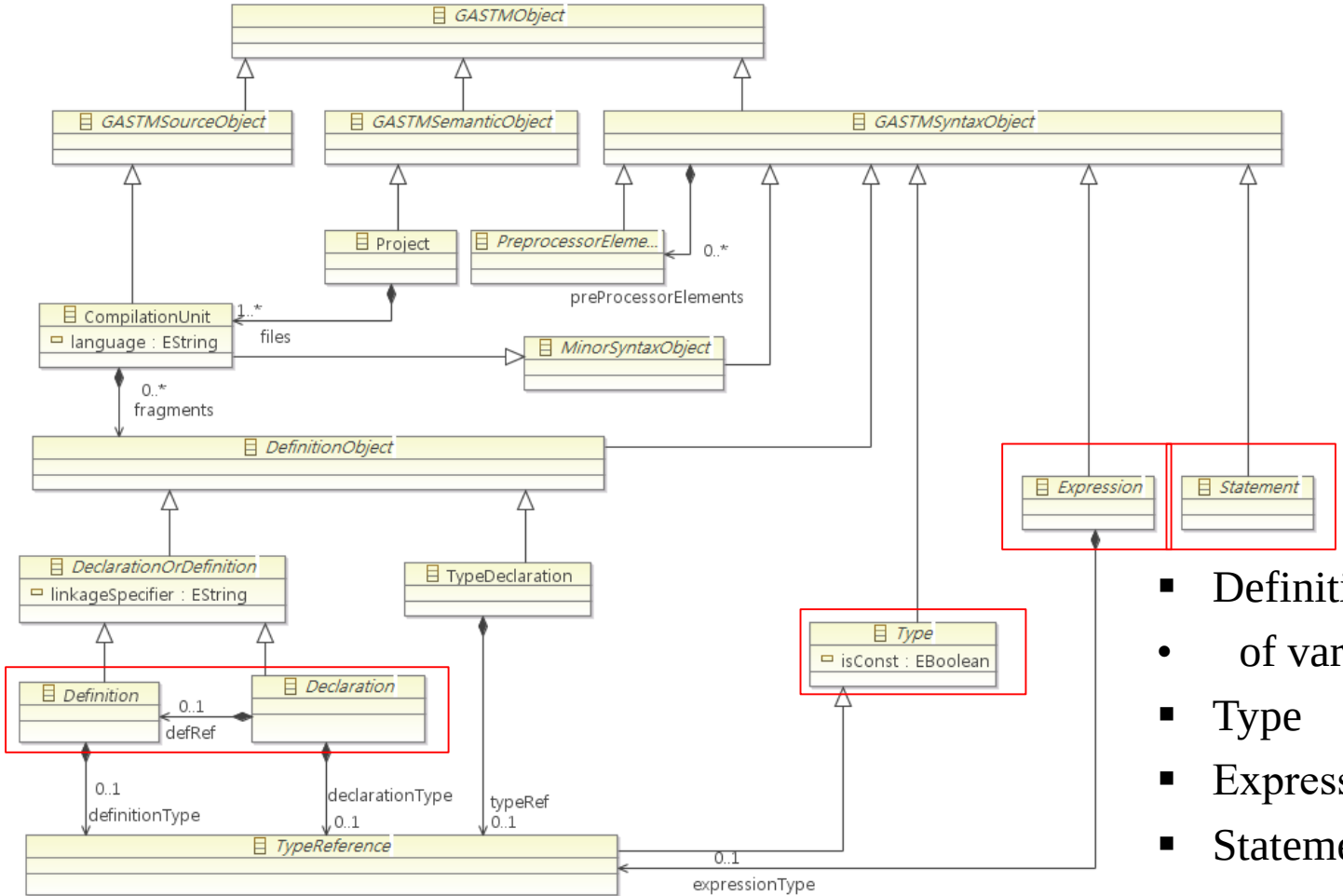
(1) Abstract Syntax Tree Metamodel(ASTM)
(2) 제안한 코드 리버스 방법
(3) 제안한 코드 생성 방법

(1) Abstract Syntax Tree Metamodel

- *Abstract Syntax Tree Metamodel (ASTM)* - OMG's 표준
  - ✓ 기존 컴파일러 들은 abstract syntax tree 가지고 있고 이를 통해 코드 생성함
  - ✓ 그러나 컴파일러 들마다 AST의 구조가 달라 이종 코드 생성의 어려움이 있음
  - ✓ ASTM은 여러 가지 언어들을 하나의 AST를 사용할 수 있도록 통합한 표준
- ASTM의 목적
  - ✓ 프로그램 코드 사이의 메타데이터를 쉽게 상호운영하기 위함
  - ✓ 예를 들어) 소프트웨어 현대화, 플랫폼, 분산 이종 환경
  - ✓ 사용 가능한 언어
    - *C, C++, C#, Java, Ada, VB/.Net, COBOL, FORTRAN, Jovial 등*
- ASTM은 표준 문서만 존재하기 때문에 구현은 또 다른 이슈임
  - ✓ ASTM의 193 엘리먼트를 전부 구현하고 부족한 부분을 추가

(1) Abstract Syntax Tree Metamodel

- Abstract Syntax Tree Metamodel (ASTM)
- GASTMObject (that is, a Root: the general ASTM)
  - ✓ Consists of SourceObject, SementicOjbect, and SystaxObject.



- Definition & Declaration
- of variable or func.
- Type
- Expression for AND, OR…
- Statements for IF, FOR

(1) Abstract Syntax Tree Metamodel 193 meta

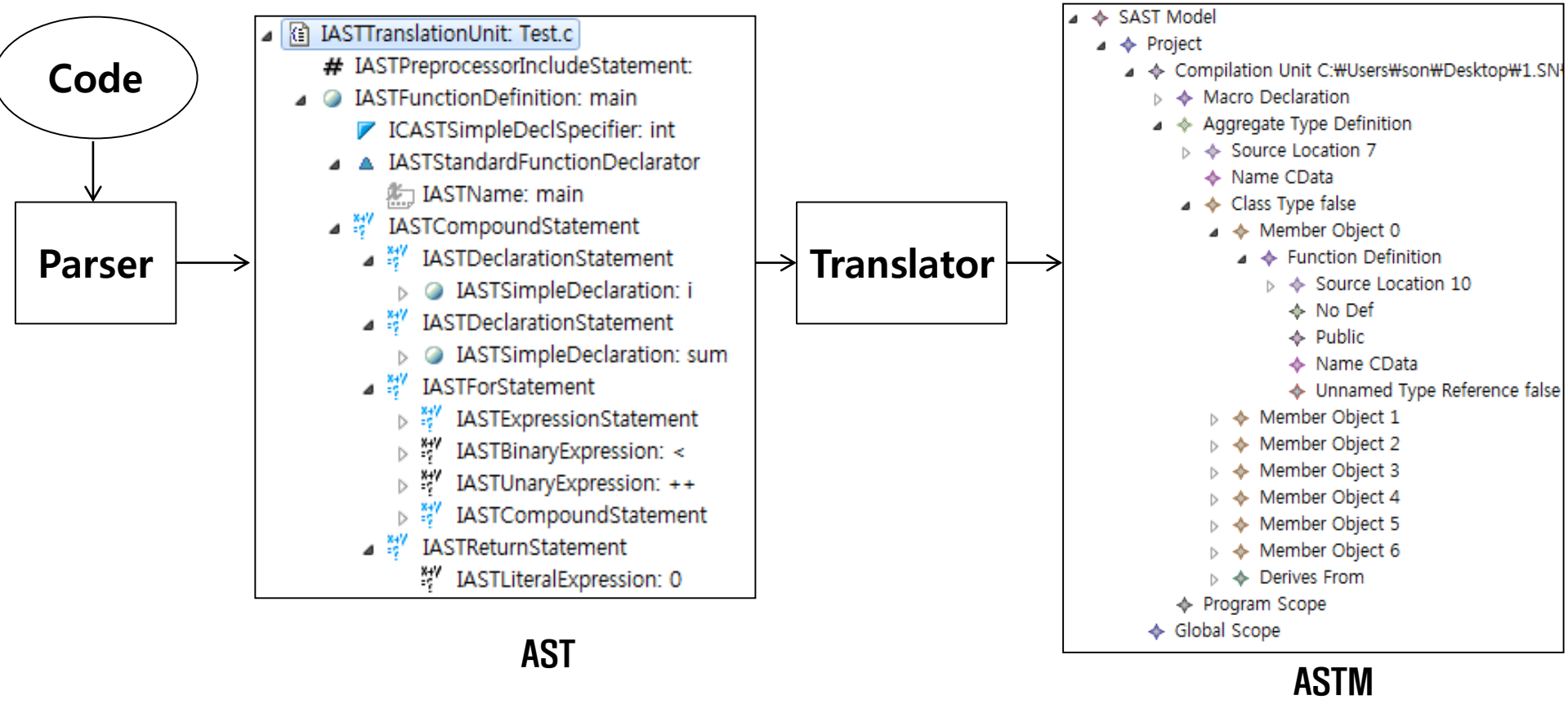
- GASTMObject
- GASTMSourceObject -> GASTMObject
- GASTMSemanticObject -> GASTMObject
- GASTMSyntaxObject -> GASTMObject
- SourceFile -> GASTMSourceObject
- SourceLocation -> GASTMSourceObject
- CompilationUnit -> SourceFile, MinorSyntaxObject
- SourceFileReference -> SourceFile
- DefinitionObject -> GASTMSyntaxObject
- ProgramScope -> Scope
- Project -> GASTMSemanticObject
- Scope -> GASTMSemanticObject
- GlobalScope -> Scope
- FunctionScope -> Scope
- AggregateScope -> Scope
- BlockScope -> Scope
- PreprocessorElement -> GASTMSyntaxObject
- AnnotationExpression -> Expression
- Type -> GASTMSyntaxObject
- Expression -> GASTMSyntaxObject
- Statement -> GASTMSyntaxObject
- MinorSyntaxObject -> GASTMSyntaxObject
- Dimension -> MinorSyntaxObject
- Name -> MinorSyntaxObject
- SwitchCase -> MinorSyntaxObject
- CatchBlock -> MinorSyntaxObject
- StorageSpecification -> MinorSyntaxObject
- VirtualSpecification -> MinorSyntaxObject
- AccessKind -> MinorSyntaxObject
- ActualParameter -> MinorSyntaxObject
- FunctionMemberAttributes -> MinorSyntaxObject
- DerivesFrom -> MinorSyntaxObject
- MemberObject -> MinorSyntaxObject
- DeclarationOrDefinition -> DefinitionObject
- TypeDefinition -> DefinitionObject
- NameSpaceDefinition -> DefinitionObject
- LabelDefinition -> DefinitionObject
- TypeDeclaration -> DefinitionObject
- Definition -> DeclarationOrDefinition
- Declaration -> DeclarationOrDefinition
- TypeReference -> Type
- FunctionDefinition -> Definition
- EntryDefinition -> Definition
- DataDefinition -> Definition
- EnumLiteralDefinition -> Definition
- FunctionDeclaration -> Declaration
- VariableDeclaration -> Declaration
- FormalParameterDeclaration -> Declaration
- External -> StorageSpecification
- FunctionPersistent -> StorageSpecification
- FileLocal -> StorageSpecification
- PerClassMember -> StorageSpecification
- NoDef -> StorageSpecification
- FormalParameterDefinition -> DataDefinition
- Virtual -> VirtualSpecification
- VariableDefinition -> DataDefinition
- BitFieldDefinition -> DataDefinition
- NamedTypeDefinition -> TypeDefinition
- AggregateTypeDefinition -> TypeDefinition
- EnumTypeDefintion -> TypeDefinition
- NamedType -> DataType
- AggregateType -> DataType
- EnumType -> DataType
- NameSpaceType -> Type
- LabelType -> Type
- AggregateTypeDeclaration -> TypeDeclaration
- EnumTypeDeclaration -> TypeDeclarati
- IncludeUnit -> PreprocessorElement
- MacroCall -> PreprocessorElement
- MacroDefinition -> PreprocessorElemer
- Comment -> PreprocessorElement
- FunctionType -> Type
- DataType -> Type
- PrimitiveType -> DataType
- ConstructedType -> DataType
- ExceptionType -> DataType
- FormalParameterType -> DataType
- NumberType -> PrimitiveType
- Void -> PrimitiveType
- Boolean -> PrimitiveType
- IntegralType -> NumberType
- RealType -> NumberType
- Byte -> NumberType
- Character -> NumberType
- ShortInteger -> IntegralType
- Integer -> IntegralType
- LongInteger -> IntegralType
- Real -> RealType
- Double -> RealType
- LongDouble -> RealType
- CollectionType -> ConstructedType
- PointerType -> ConstructedType
- ReferenceType -> ConstructedType
- RangeType -> ConstructedType
- ArrayType -> ConstructedType
- StructureType -> AggregateType
- UnionType -> AggregateType
- ClassType -> AggregateType
- AnnotationType -> AggregateType

(1) Abstract Syntax Tree Metamodel 193 meta

- ByValueFormalParameterType -> FormalParameterType
- ByReferenceFormalParameterType -> FormalParameterType
- Public -> AccessKind
- Protected -> AccessKind
- Private -> AccessKind
- UnnamedTypeReference -> TypeReference
- NamedTypeReference -> TypeReference
- ExpressionStatement -> Statement
- JumpStatement -> Statement
- BreakStatement -> Statement
- ContinueStatement -> Statement
- LabeledStatement -> Statement
- BlockStatement -> Statement
- EmptyStatement -> Statement
- IfStatement -> Statement
- SwitchStatement -> Statement
- ReturnStatement -> Statement
- LoopStatement -> Statement
- TryStatement -> Statement
- DeclarationOrDefinitionStatement -> Statement
- ThrowStatement -> Statement
- DeleteStatement -> Statement
- TerminateStatement -> Statement
- LabelAccess -> Expression
- CaseBlock -> SwitchCase
- DefaultBlock -> SwitchCase
- WhileStatement -> LoopStatement
- DoWhileStatement -> LoopStatement
- ForStatement -> LoopStatement
- ForCheckBeforeStatement -> ForStatement
- ForCheckAfterStatement -> ForStatement
- TypesCatchBlock -> CatchBlock
- VariableCatchBlock -> CatchBlock
- Literal -> Expression
- CastExpression -> Expression
- AggregateExpression -> Expression
- BinaryExpression -> Expression
- ConditionalExpression -> Expression
- RangeExpression -> Expression
- FunctionCallExpression -> Expression
- NewExpression -> Expression
- NameReference -> Expression
- ArrayAccess -> Expression
- CollectionExpression -> Expression
- IdentifierRefrerence -> NameReference
- QualifiedIdentifierReference -> NameReference
- TypeQualifiedIdentifierReference -> NameReference
- QualifiedOverPointer -> QualifiedIdentifierReference
- QualifiedOverData -> QualifiedIdentifierReference
- IntegerLiteral -> Literal
- StringLiteral -> Literal
- CharLiteral -> Literal
- RealLiteral -> Literal
- BooleanLiteral -> Literal
- BitLiteral -> Literal
- EnumLiteral -> Literal
- UnaryExpression -> Expression
- UnaryOperator -> MinorSyntaxObject
- UnaryPlus -> UnaryOperator
- UnaryMinus -> UnaryOperator
- Not -> UnaryOperator
- BitNot -> UnaryOperator
- Decrement -> UnaryOperator
- PostIncrement -> UnaryOperator
- PostDecrement -> UnaryOperator
- BinaryOperator -> MinorSyntaxObject
- Add -> BinaryOperator
- Subtract -> BinaryOperator
- Multiply -> BinaryOperator
- Divide -> BinaryOperator
- Modulus -> BinaryOperator
- Exponent -> BinaryOperator
- And -> BinaryOperator
- Or -> BinaryOperator
- Equal -> BinaryOperator
- NotEqual -> BinaryOperator
- Greater -> BinaryOperator
- NotGreater -> BinaryOperator
- Less -> BinaryOperator
- NotLess -> BinaryOperator
- BitAnd -> BinaryOperator
- BitOr -> BinaryOperator
- BitXor -> BinaryOperator
- BitLeftShift -> BinaryOperator
- BitRightShift -> BinaryOperator
- Assign -> BinaryOperator
- OperatorAssign -> BinaryOperator
- ActualParameterExpression -> ActualPara
- MissingActualParameter -> ActualParam
- ByValueActualParameterExrpession -> Ac
- ByReferenceActualParameterExrpession -
- Increment -> UnaryOperator
- AddressOf -> UnaryOperator
- Deref -> UnaryOperator

(2) 제안한 코드 리버스 방법

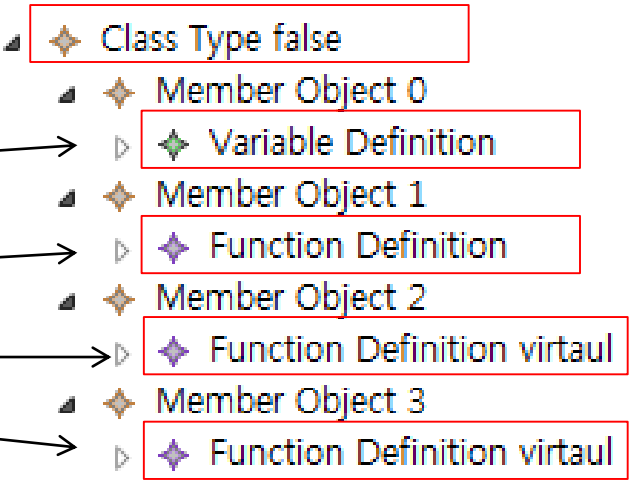
- 파서 : 코드를 AST로 변환
- 변환기 : AST를 ASTM으로 변환



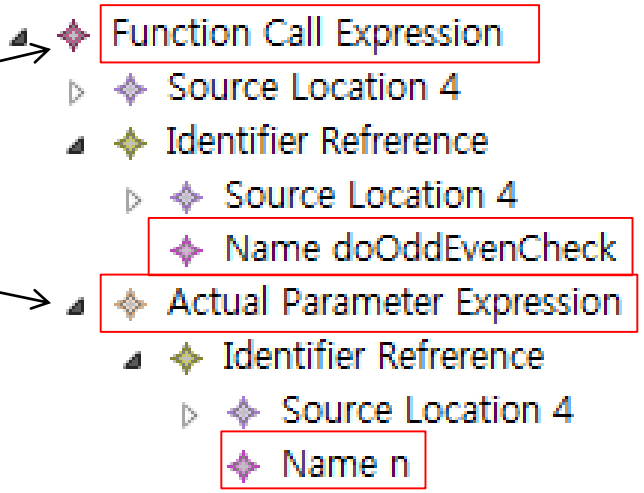
(2) 제안한 코드 리버스 방법

➢ Class definition & function call statement

```
class Figure {
public :
        int x;
        Figure();
        virtual ~Figure();
        virtual int getArea();
};
```



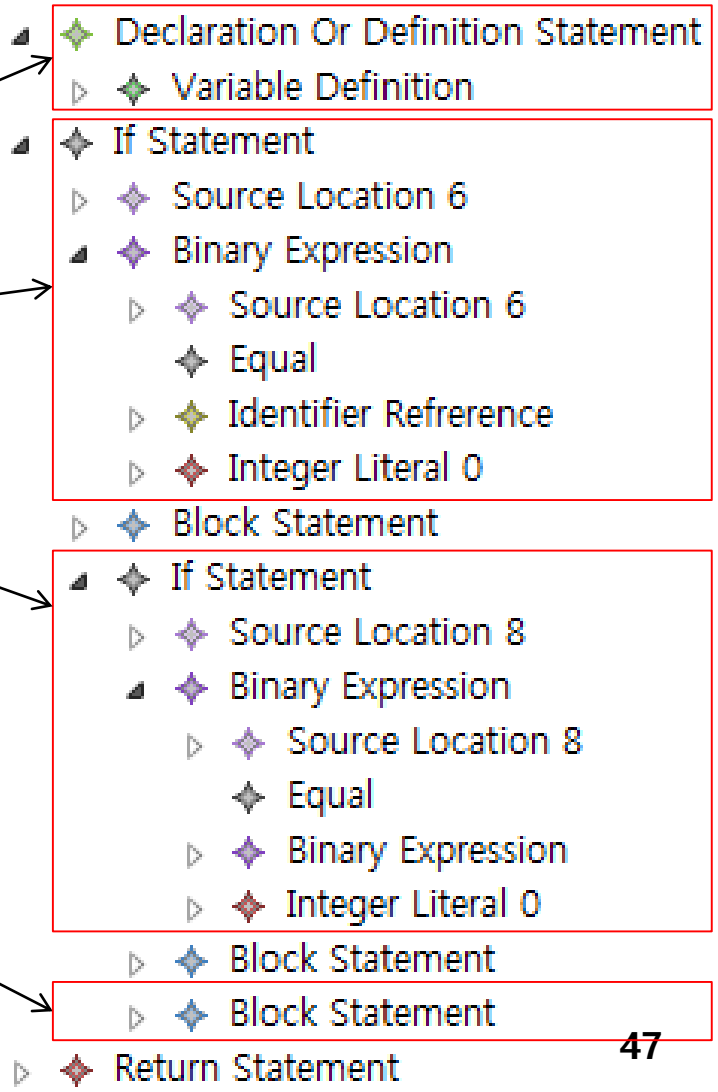

```
int main() {
        int n = 7;
        doOddEvenCheck(n);
        return 0;
}
```



(2) 제안한 코드 리버스 방법

➢ Variable definition & if statement

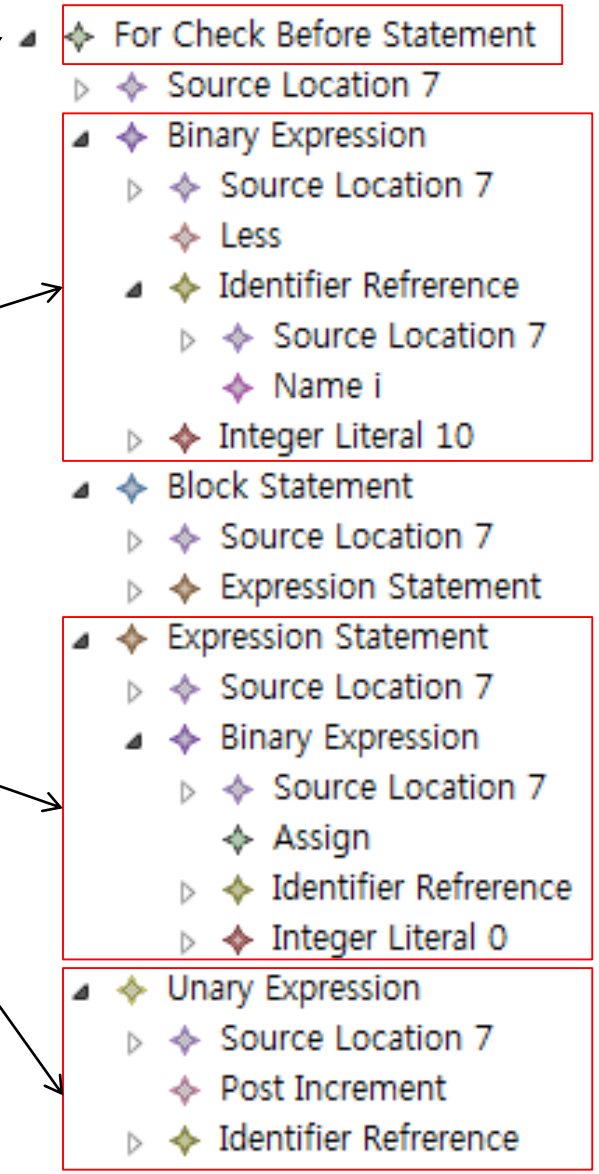
```
#include "stdio.h"
int main() {
        int n = 10;
        if(n == 0) {
                printf("zero");
        } else if(n % 2 == 0) {
                printf("even number");
        } else {
                printf("odd number");
        }
        return 0;
}
```



(2) 제안한 코드 리버스 방법

- For statement

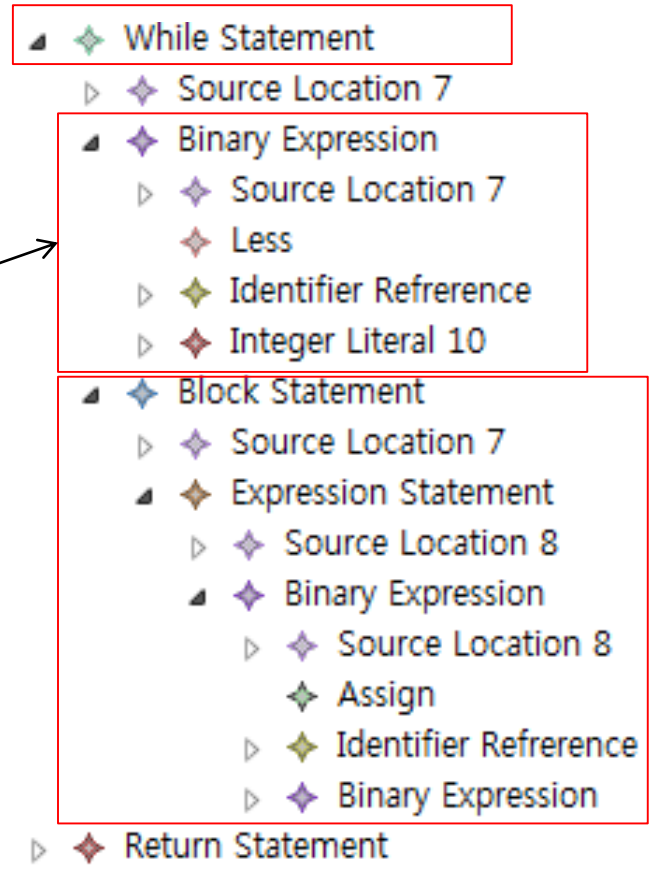
```
#include "stdio.h"
int main() {
        int i;
        int sum = 0;
        for(i = 0; i < 10; i++) {
                sum = sum + i;
        }
        return 0;
}
```



(2) 제안한 코드 리버스 방법

➢ While statement

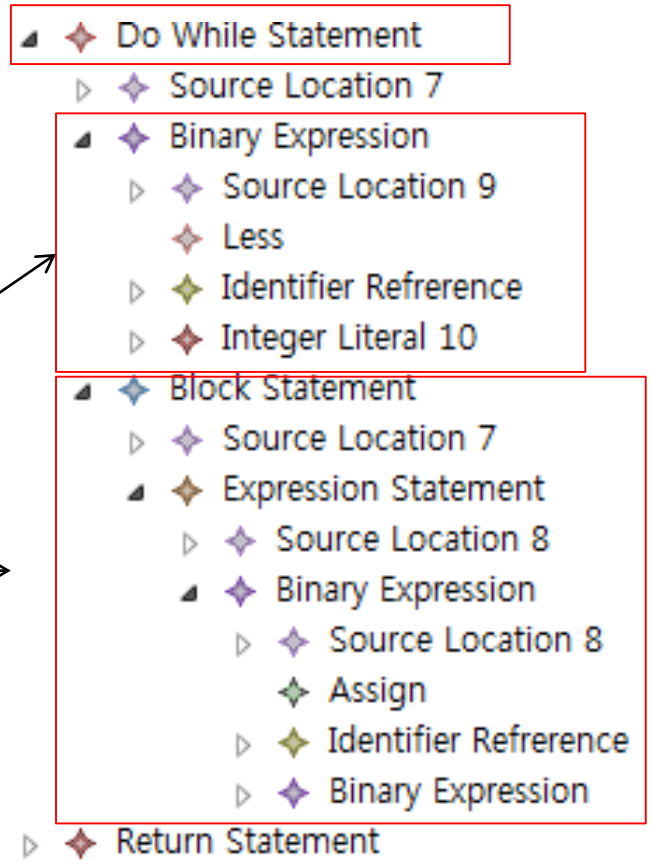
```
#include "stdio.h"
int main() {
        int i = 0;
        int sum = 0;
        while(i < 10) {
                sum = sum + i;
        }
        return 0;
}
```



(2) 제안한 코드 리버스 방법

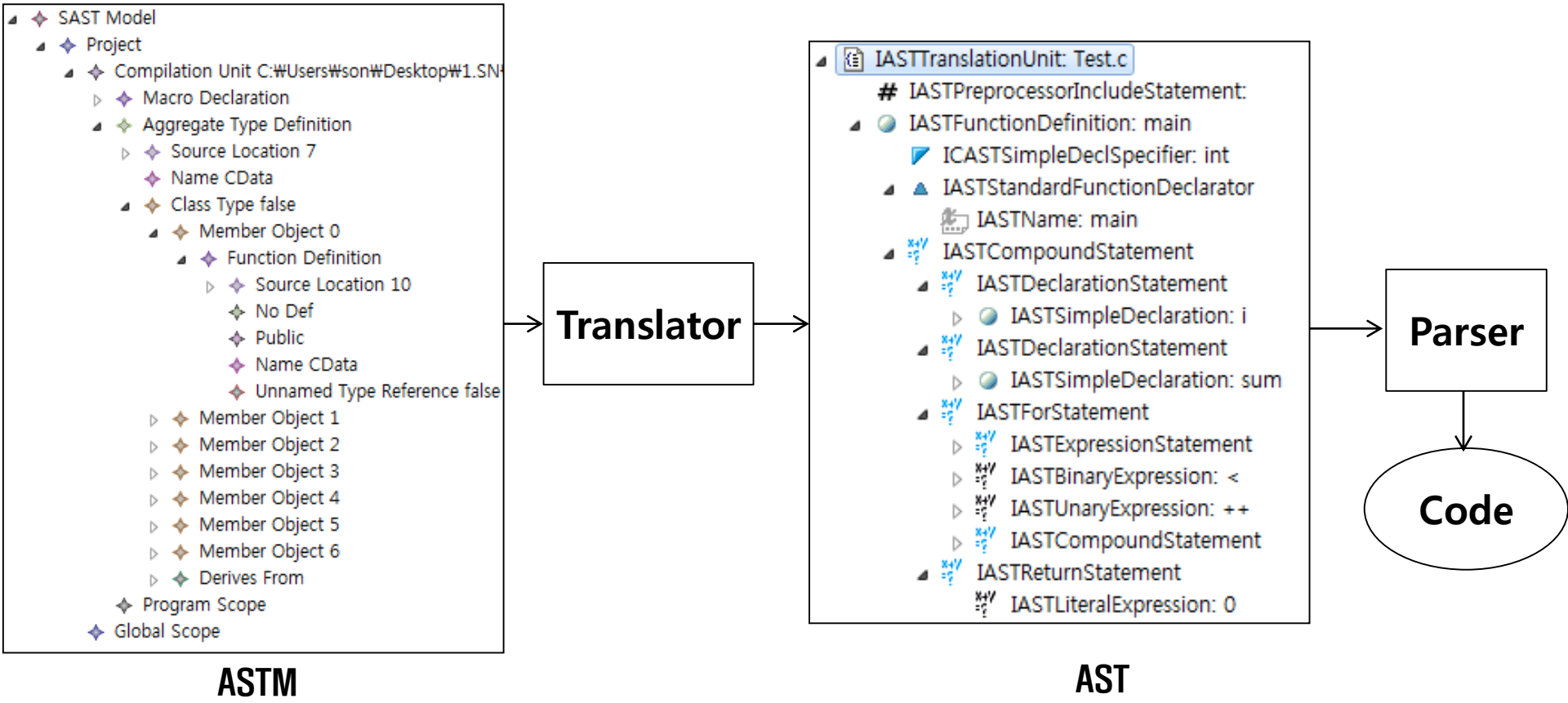
➢ Do while statement

```
#include "stdio.h"
int main() {
        int i = 0;
        int sum = 0;
        do {
                sum = sum + i;
        } while(i < 10);
        return 0;
}
```



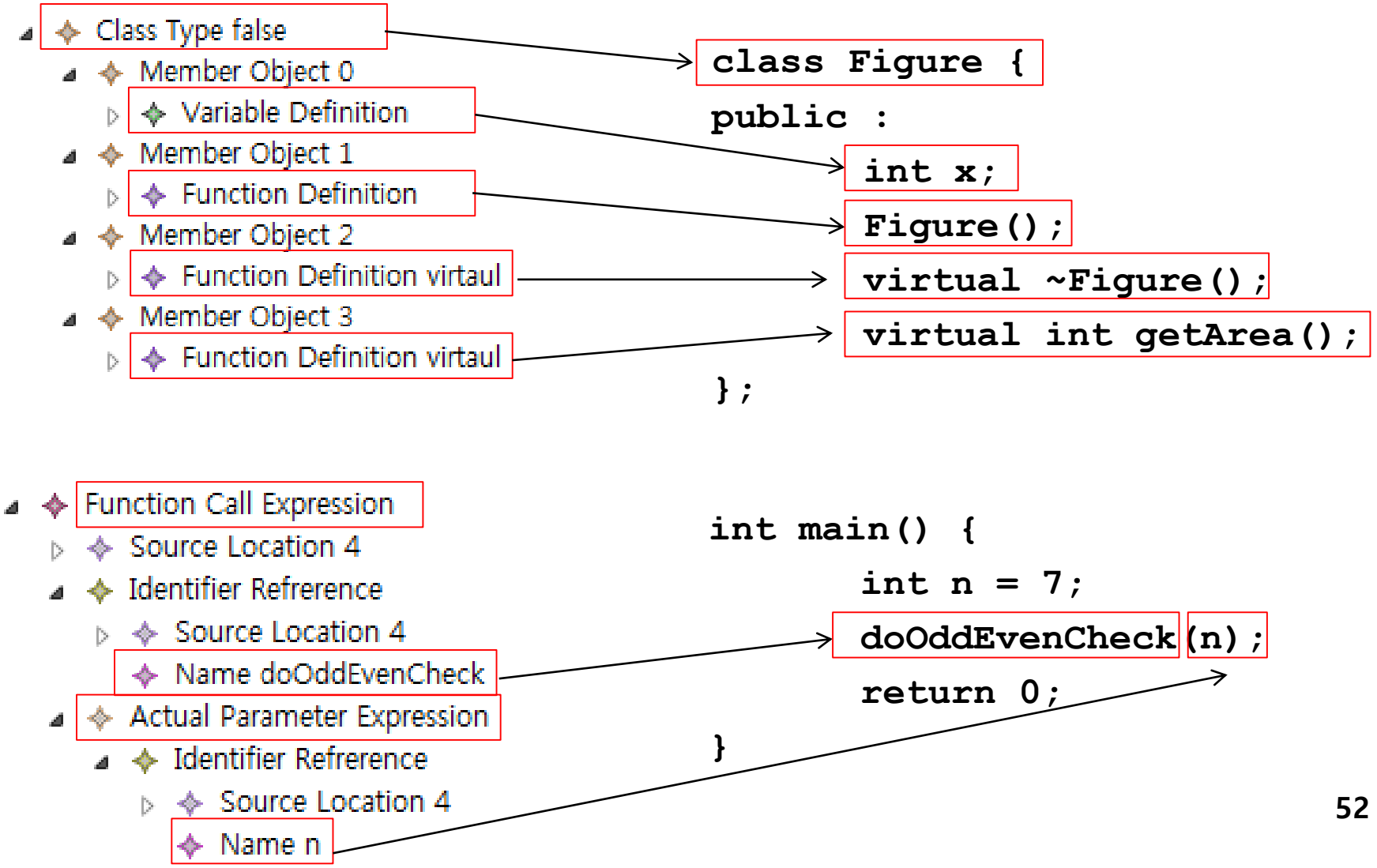
(3) 제안한 코드 생성 방법

- 변환기 : ASTM를 AST로 변환
- 파서 : AST로 코드 생성



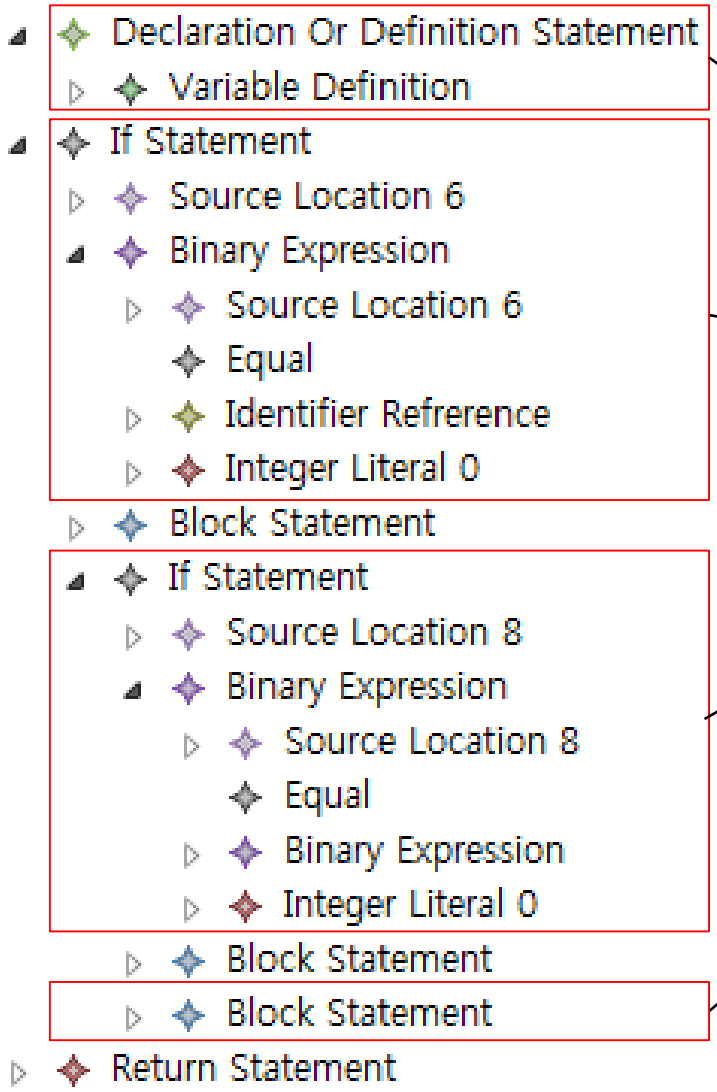
(3) 제안한 코드 생성 방법

➢ Class definition & function call statement



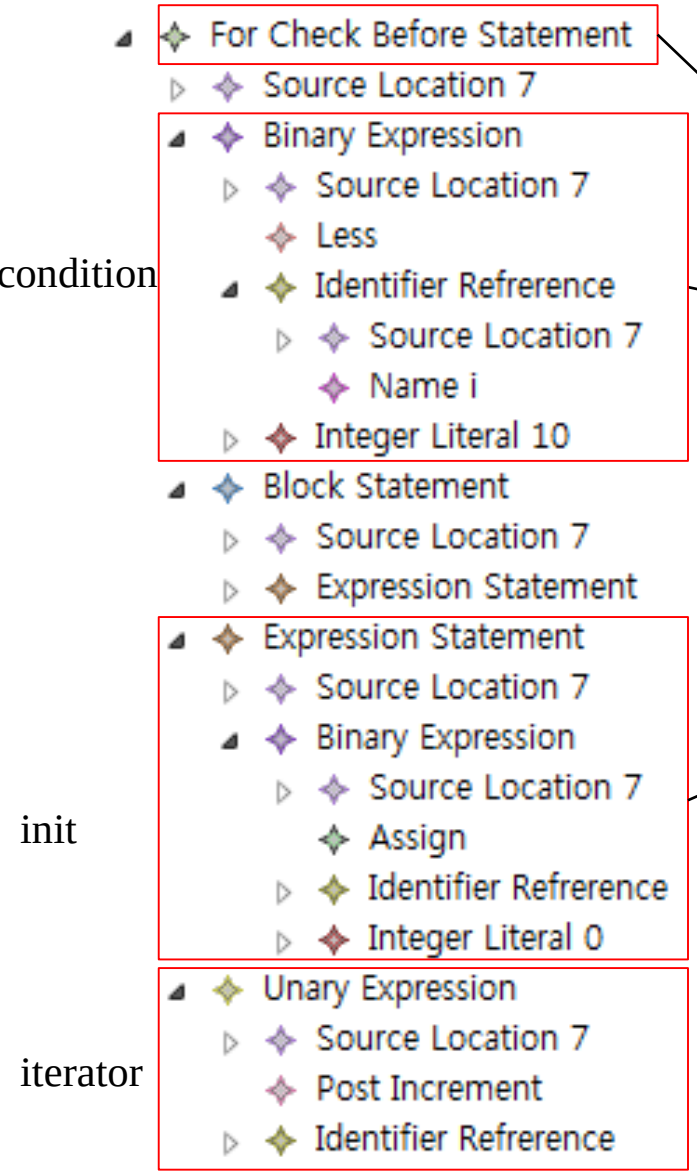
(3) 제안한 코드 생성 방법

➢ Variable definition & if statement




```
#include "stdio.h"
int main() {
    int n = 10;
    if(n == 0) {
        printf("zero");
    } else if(n % 2 == 0) {
        printf("even number");
    } else {
        printf("odd number");
    }
    return 0;
}
```

(3) 제안한 코드 생성 방법

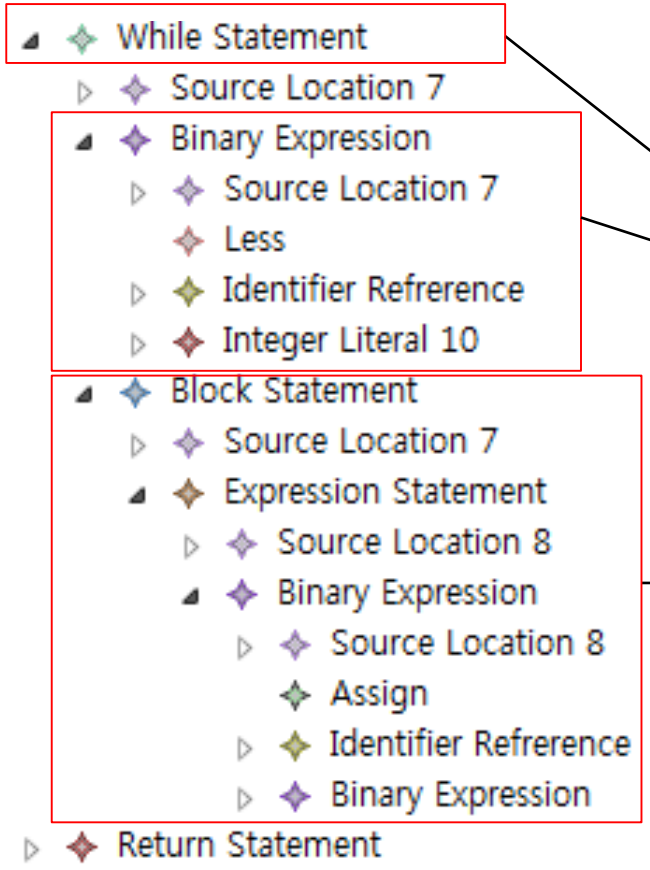


For statement

```
#include "stdio.h"
int main() {
        int i;
        int sum = 0;
        for(i = 0; i < 10; i++) {
                sum = sum + i;
        }
        return 0;
}
```

(3) 제안한 코드 리버스 방법

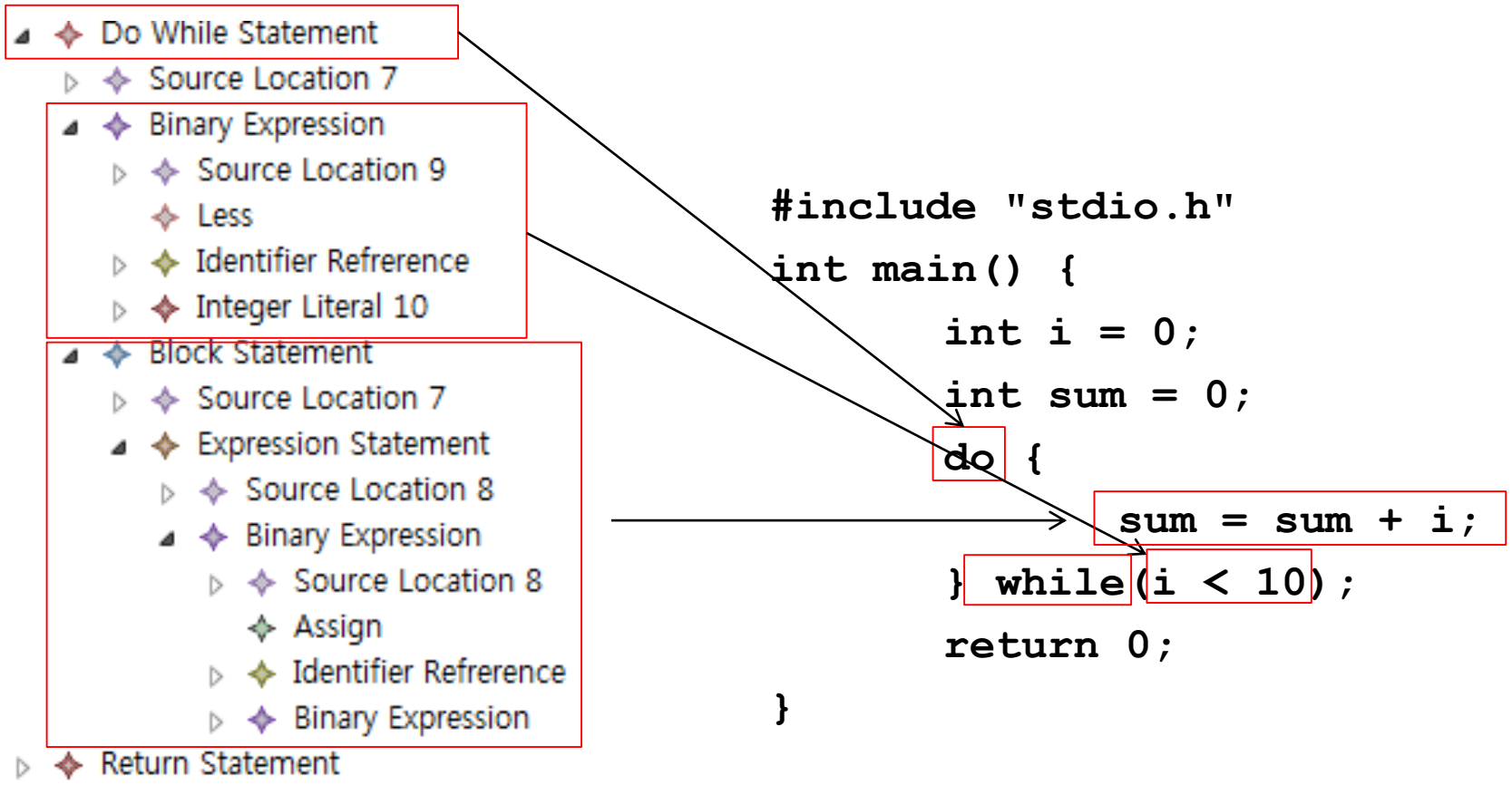
➢ While statement




```
#include "stdio.h"
int main() {
        int i = 0;
        int sum = 0;
        while(i < 10) {
                sum = sum + i;
        }
        return 0;
}
```

(3) 제안한 코드 리버스 방법

➢ Do while statement



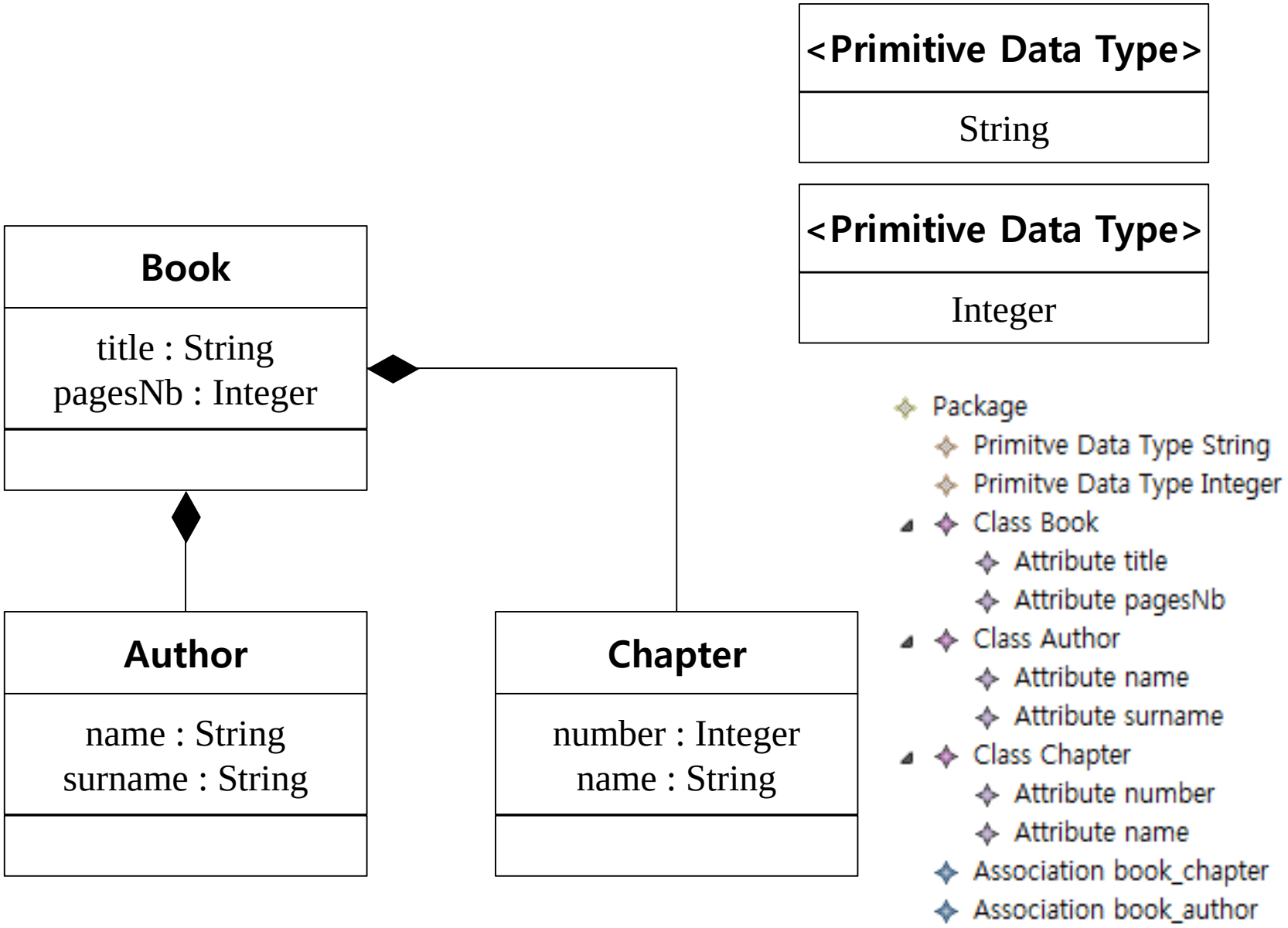
4. 적용 사례

1) 양방향 모델 변환
2) 양방향 코드 변환

1) 양방향 모델 변환

- 모델변환 예
  - ✓ UML 클래스 다이어그램 → RDBMS

Simple UML Class Diagram

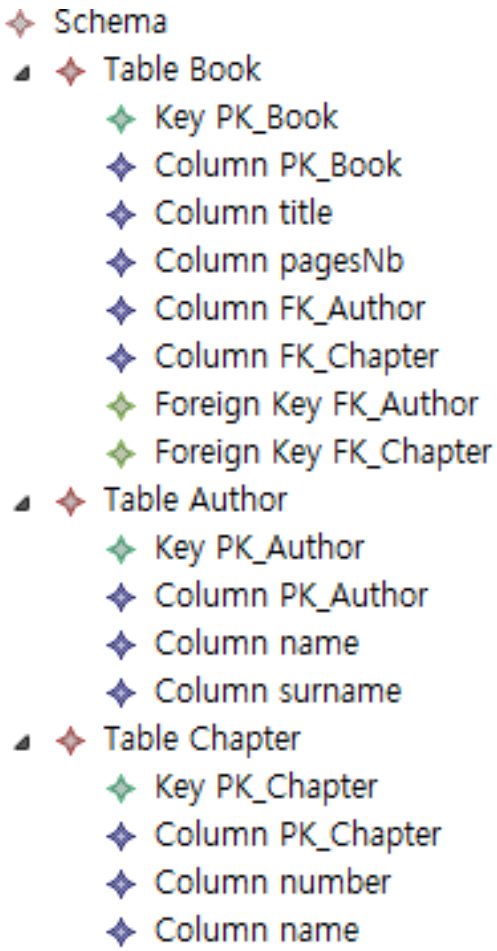


Simple RDBMS

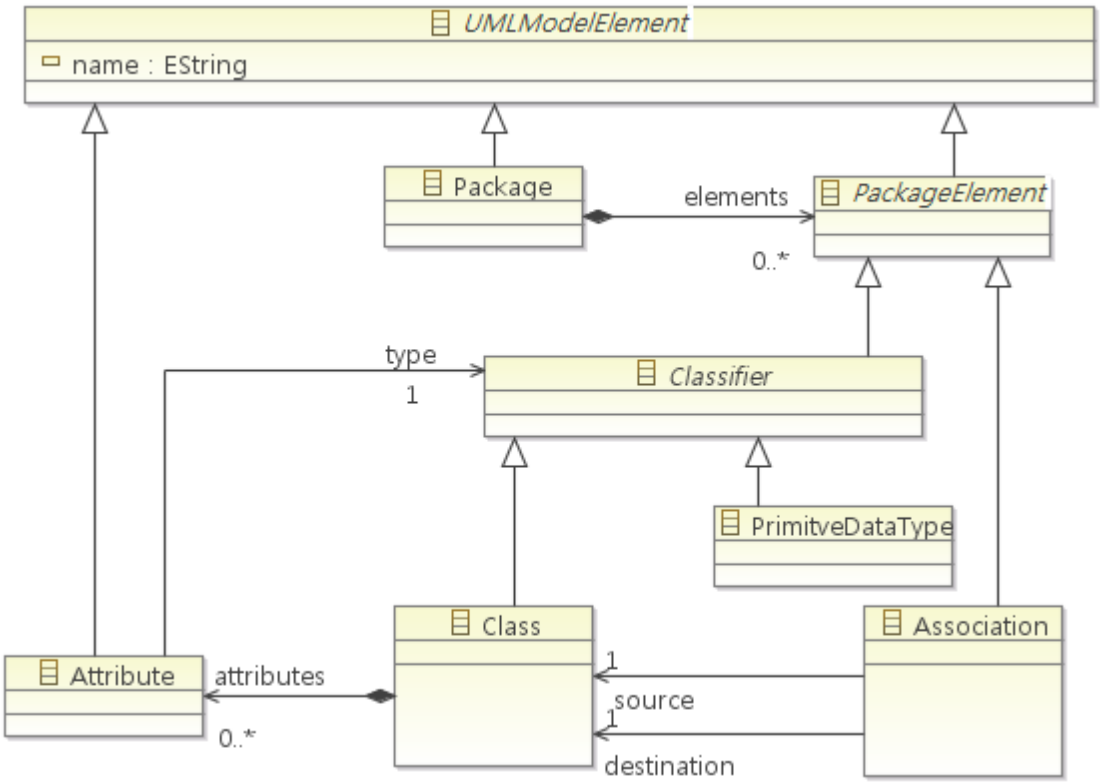
Book		
PK	PK_book	number
	title	varchar
	pagesNb	number
FK	FK_author	number
FK	FK_chapter	number

Author		
PK	PK_author	number
	name	varchar
	surname	varchar

Chapter		
PK	PK_chapter	number
	number	number
	name	varchar

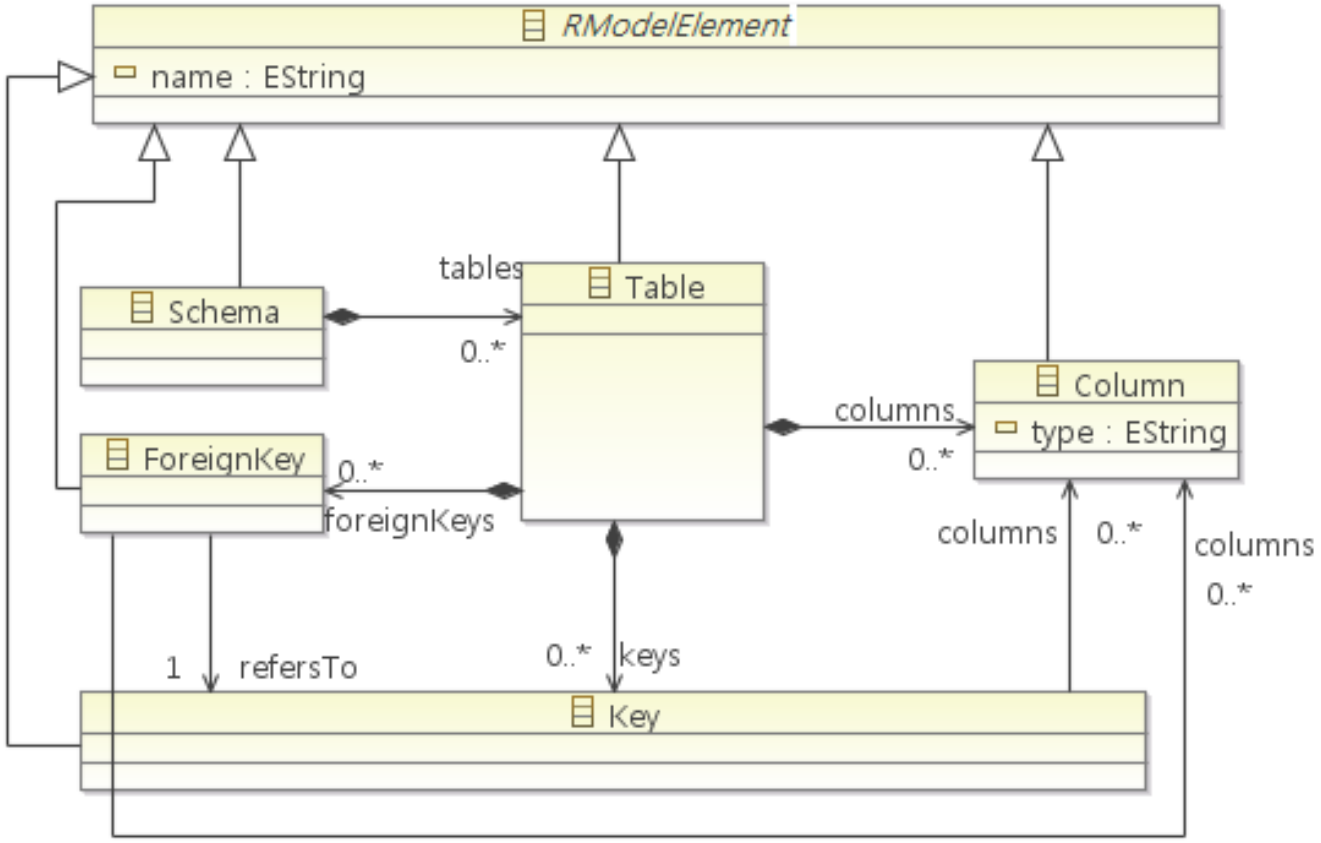


Simple UML 클래스 다이어그램 메타모델



- simpleumlmetamodel
  - :Package(name:EString)
    - [0..*]elements:Class(name:EString)
      - [0..*]attributes:Attribute(name:EString, type:Class | PrimitveDataType)
    - [0..*]elements:PrimitveDataType(name:EString)
    - [0..*]elements:Association(name:EString, source:Class, destination:Class)

Simple RDBMS 메타모델



- simplerdbmsmetamodel
  - :Schema(name:EString)
    - [0..*]tables:Table(name:EString)
      - [0..*]keys:Key(name:EString, [0..*]columns:Column)
      - [0..*]columns:Column(name:EString, type:EString)
      - [0..*]foreignKeys:ForeignKey(name:EString, refersTo:Key, [0..*]columns:Column)

Rule Package <-> Schema

t1 : name <-> name

t1

Package(name:EString)

[0..*]PrimitveDataType(name:EString)

[0..*]Class(name:EString)

[0..*]Attribute(name:EString, type:PrimitveDataType|Class)

[0..*]Association(name:EString, source:Class, desitination:Class)

Schema(name:EString)

[0..*]Table(name:EString)

[0..*]Column(name:EString, type:EString)

[0..*]Key(name:EString, [0..*]columns:Column)

[0..*]ForeignKey(name:EString, refersTo:Key, [0..*]columns:Colum

Rule Class <–> Table

t1 : name <-> name

t2 : ? <-> 'number'

t3 : name <-> 'PK_'+name

모든 룰은 없으면 생성
있으면 참조

Package(name:EString)

[0..*]PrimitveDataType(name:EString)

[0..*]Class(name:EString)

[0..*]Attribute(name:EString, type:PrimitveDataType|Class)

[0..*]Association(name:EString, source:Class, desitination:Class)

t1

Schema(name:EString)

[0..*]Table(name:EString)

t1

t2

[0..*]Column(name:EString, type:EString)

t3

[0..*]Key(name:EString, [0..*]columns:Column)

[0..*]ForeignKey(name:EString, refersTo:Key, [0..*]columns:Colum

Rule Attribute <–> Column

t1 : name <-> name

t2 : name <-> type
'Integer' <iff> 'number'
'String' <iff> 'varcahr'

Package(name:EString)

[0..*]PrimitveDataType(name:EString)

[0..*]Class(name:EString)

[0..*]Attribute(name:EString, type:PrimitveDataType|Class)

[0..*]Association(name:EString, source:Class, desitination:Class)

Schema(name:EString)

[0..*]Table(name:EString)

[0..*]Column(name:EString, type:EString)

[0..*]Key(name:EString, [0..*]columns:Column)

[0..*]ForeignKey(name:EString, refersTo:Key, [0..*]columns:Colum

t1

t2

Rule Association <-> ForeignKey

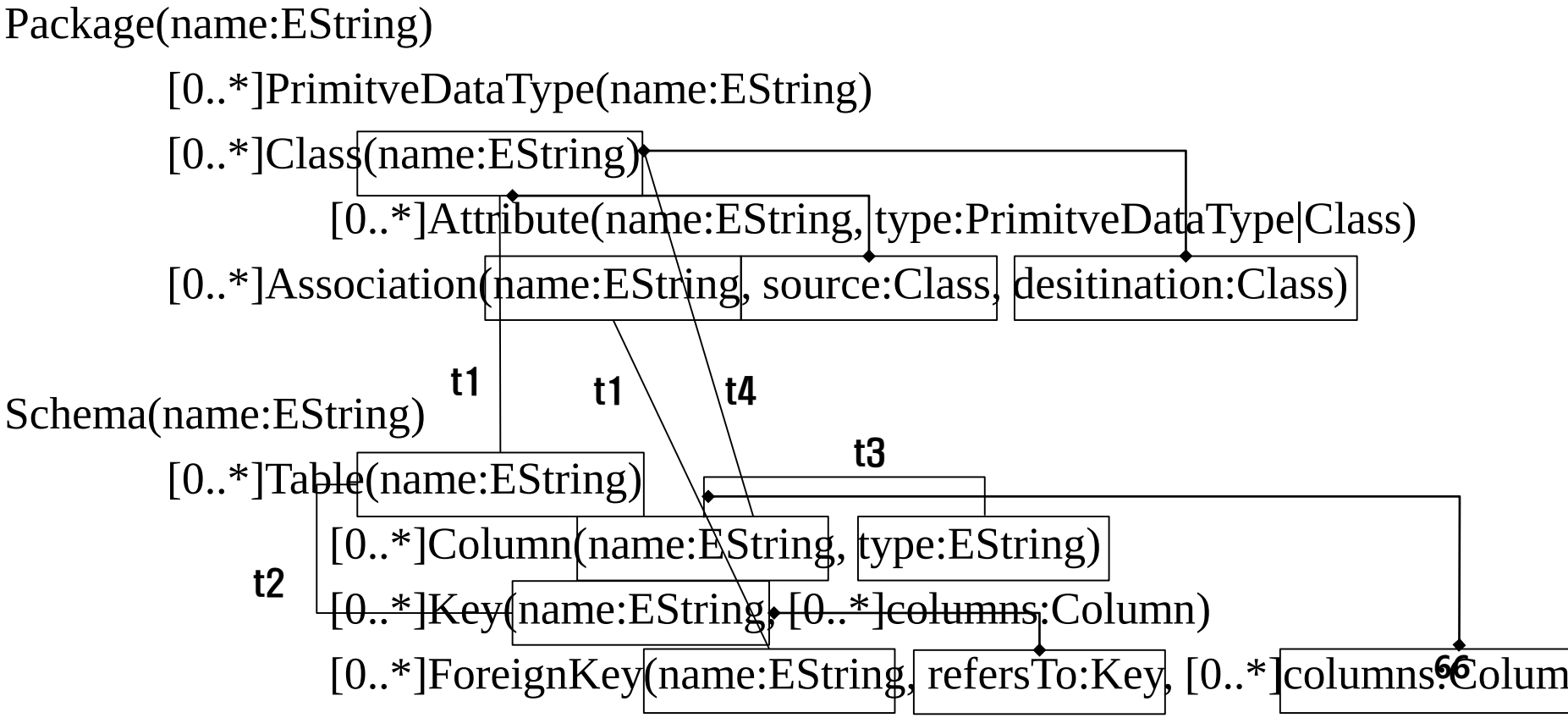
t1 : name <-> name

t3 : ? <-> 'number'

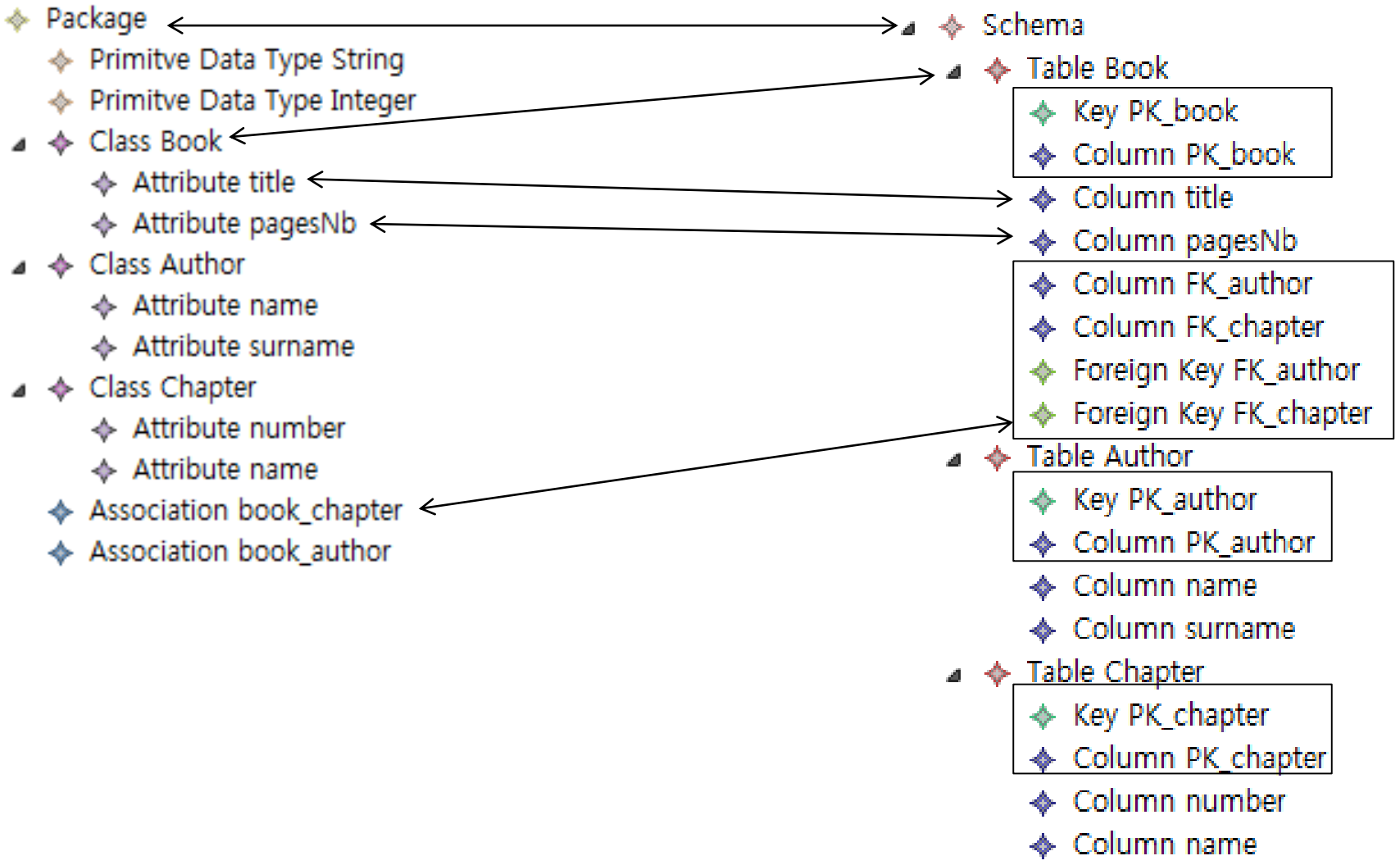
t2 : name <-> 'PK_'+name

t4 : name <-> 'FK_'+name

모든 룰은 없으면 생성
있으면 참조



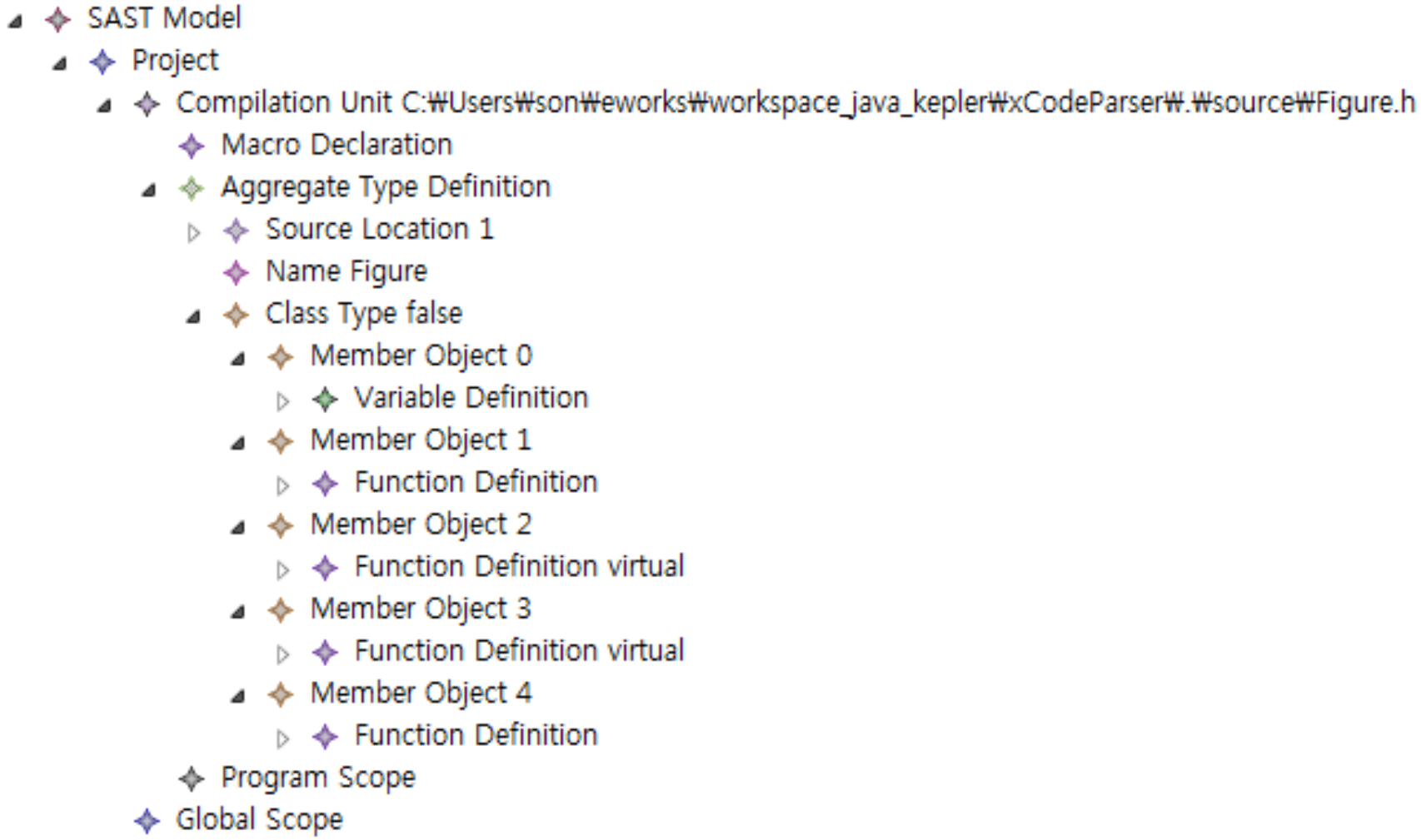
변환 결과


Package
Primitve Data Type String
Primitve Data Type Integer
Class Book
Attribute title
Attribute pagesNb
Class Author
Attribute name
Attribute surname
Class Chapter
Attribute number
Attribute name
Association book_chapter
Association book_author
Schema
Table Book
Key PK_book
Column PK_book
Column title
Column pagesNb
Column FK_author
Column FK_chapter
Foreign Key FK_author
Foreign Key FK_chapter
Table Author
Key PK_author
Column PK_author
Column name
Column surname
Table Chapter
Key PK_chapter
Column PK_chapter
Column number
Column name

2) 양방향 코드 변환

- 코드 생성
  - ✓ ASTM→ C/C++
  - ✓ ASTM→ Java
- 코드의 역변환
  - ✓ C/C++ → Graph
  - ✓ Java → Graph

ASTM


SAST Model
Project
Compilation Unit C:₩Users₩son₩eworks₩workspace_java_kepler₩xCodeParser₩.₩source₩Figure.h
Macro Declaration
Aggregate Type Definition
Source Location 1
Name Figure
Class Type false
Member Object 0
Variable Definition
Member Object 1
Function Definition
Member Object 2
Function Definition virtual
Member Object 3
Function Definition virtual
Member Object 4
Function Definition
Program Scope
Global Scope

생성 코드 비교

```
class Figure
{
private:
        int x;

public :
        Figure();
        virtual ~Figure();
        virtual int getArea();
        void doOddEvenCheck(int n);
};
```

원본 코드

```
class Figure
{
private:
    int x;

public:
    Figure();
    virtual ~Figure();
    virtual int getArea();
    void doOddEvenCheck(int n);
};
```

생성 코드

```
Figure::Figure()
{
}
Figure::~Figure()
{
}
void Figure::doOddEvenCheck(int n)
{
   int n = 10;
   if(n == 0) {
      printf("zero");
   } else if(n % 2 == 0) {
      printf("even number");
   } else {
      printf("odd number");
   }
}


int Figure::getArea()
{
   int n = 7;
   doOddEvenCheck(n);
   int i;
   int sum = 0;
   for(i = 0; i < 10; i++) {
      sum = sum + i;
   }
   return sum;
}
```

```
Figure::Figure()
{
}
Figure::~Figure()
{
}
void Figure::doOddEvenCheck(int n)
{
    int n = 10;
    if (n == 0){
        printf("zero");
    }else{
        if (n % 2 == 0){
            printf("even number");
        }else{
            printf("odd number");
        }
    }
}

int Figure::getArea()
{
    int n = 7;
    doOddEvenCheck(n);
    int i;
    int sum = 0;
    for (i = 0;i < 10;i++){
        sum = sum + i;
    }
    return sum;
}
```

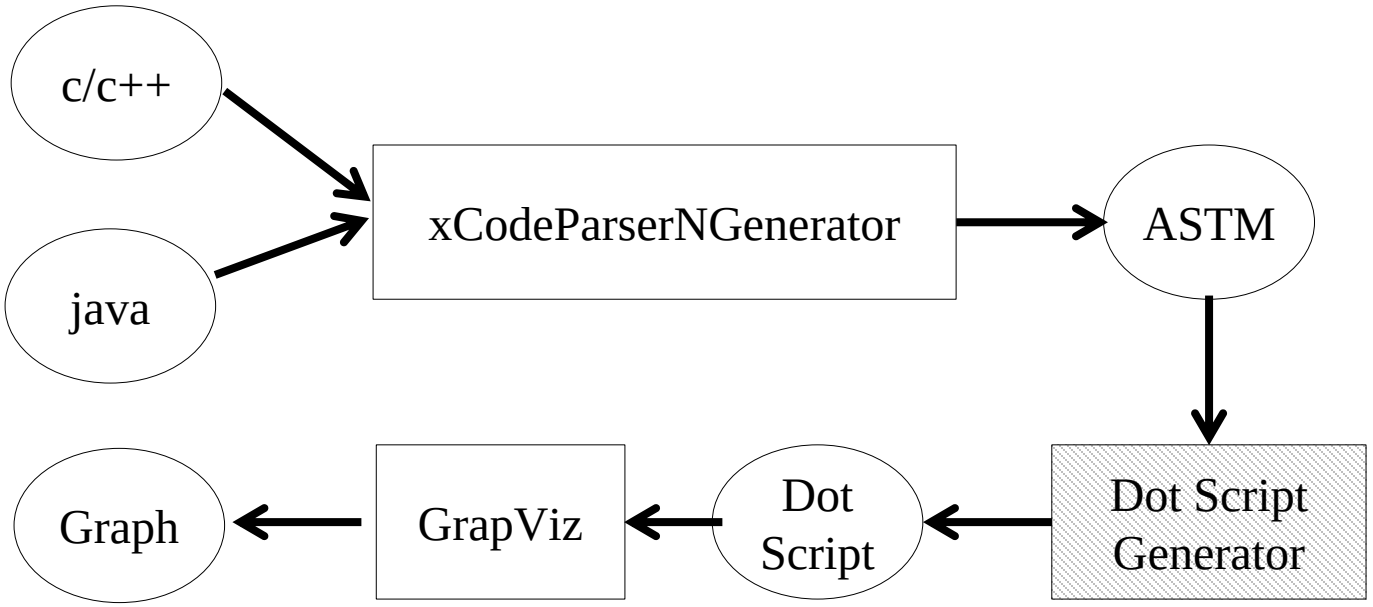
```
class Figure {
private:
   int x;
public :
   Figure();
   virtual ~Figure();
   virtual int getArea();
   void doOddEvenCheck(int n);
}
Figure::Figure() {}
Figure::~Figure() {}
void Figure::doOddEvenCheck(int n){
   int n = 10;
   if(n == 0) {
      printf("zero");
   } else if(n % 2 == 0) {
      printf("even number");
   } else {
      printf("odd number");
   }
}
int Figure::getArea() {
   int n = 7;
   doOddEvenCheck(n);
   int i;
   int sum = 0;
   for(i = 0; i < 10; i++) {
      sum = sum + i;
   }
   return sum;
}
```

```
public class Figure {
   private int x;
   public void Figure() {
   }
   public void closeFigure();
   public void doOddEvenCheck(int n){
      int n = 10;
      if (n == 0) {
         printf("zero");
      } else {
         if (n % 2 == 0) {
            printf("even number");
         } else {
            printf("odd number");
         }
      }
   }

   public int getArea() {
      int n = 7;
      doOddEvenCheck(n);
      int i;
      int sum = 0;
      for (i = 0; i < 10; i++) {
         sum = sum + i;
      }
      return sum;
   }
}
```

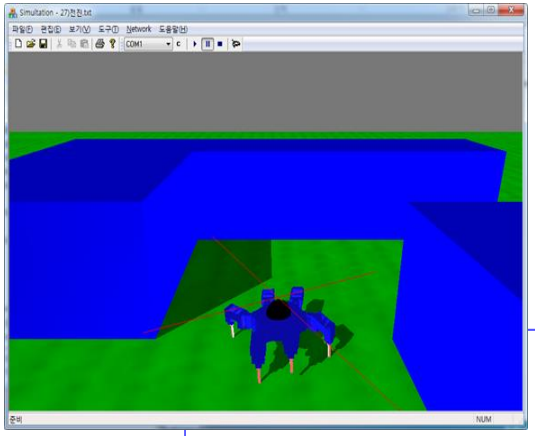
Java 생성 코드

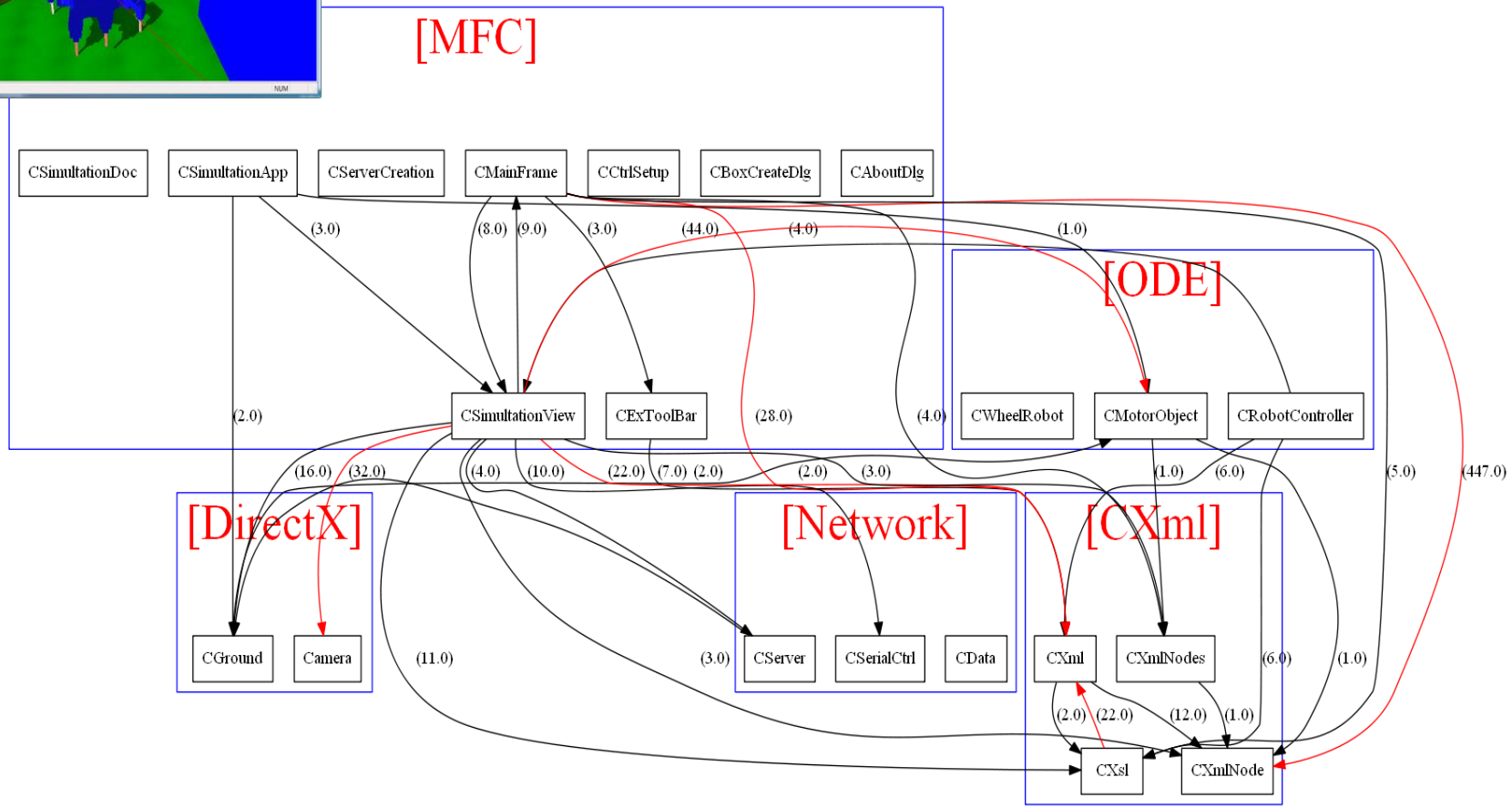
코드의 역변환

- SW Visualization과 접목

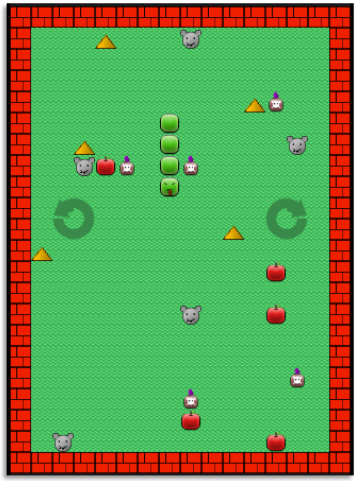


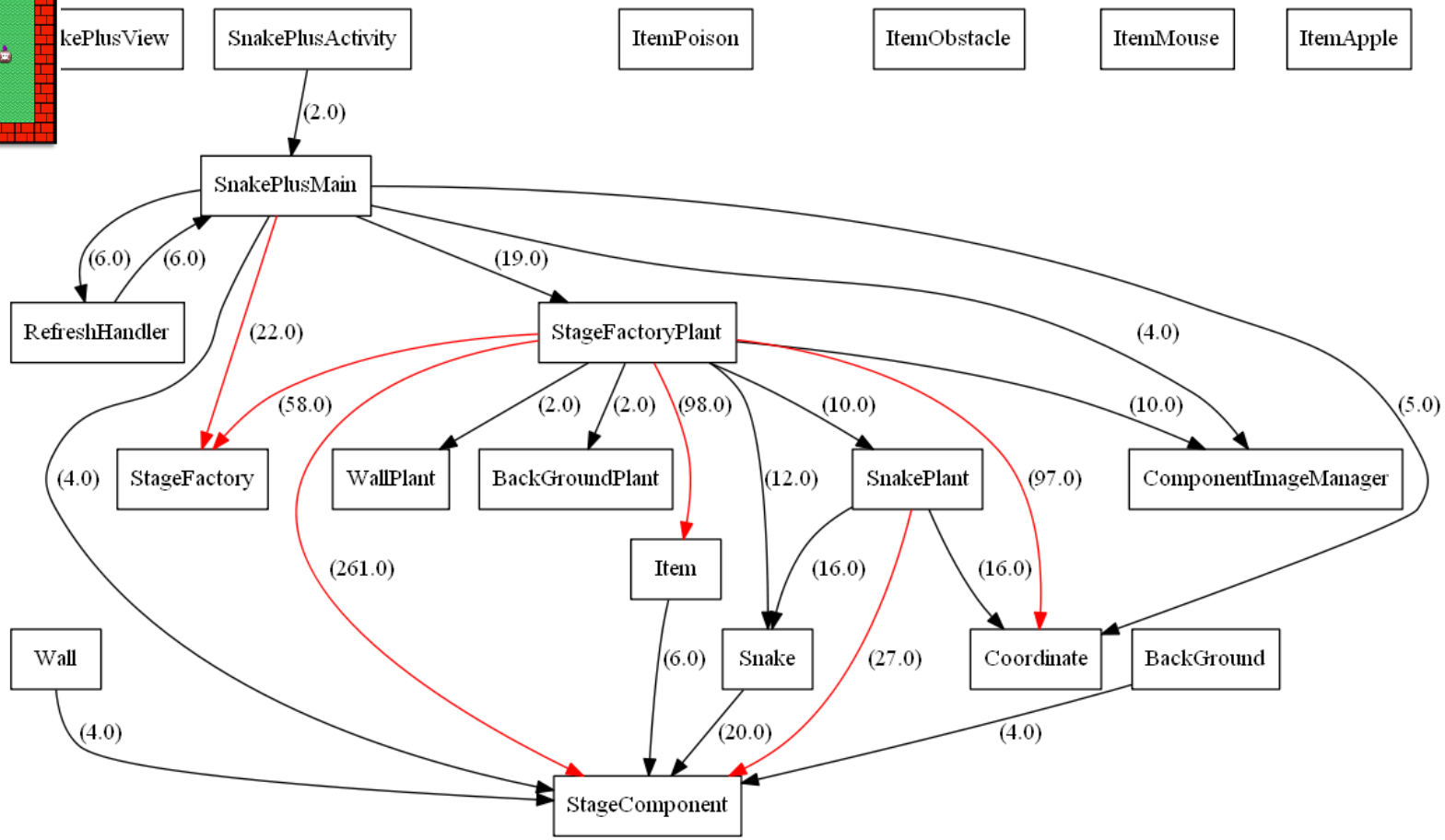
다관절 로봇 시뮬레이터 적용(C/C++)





스네이크 플러스(Java)



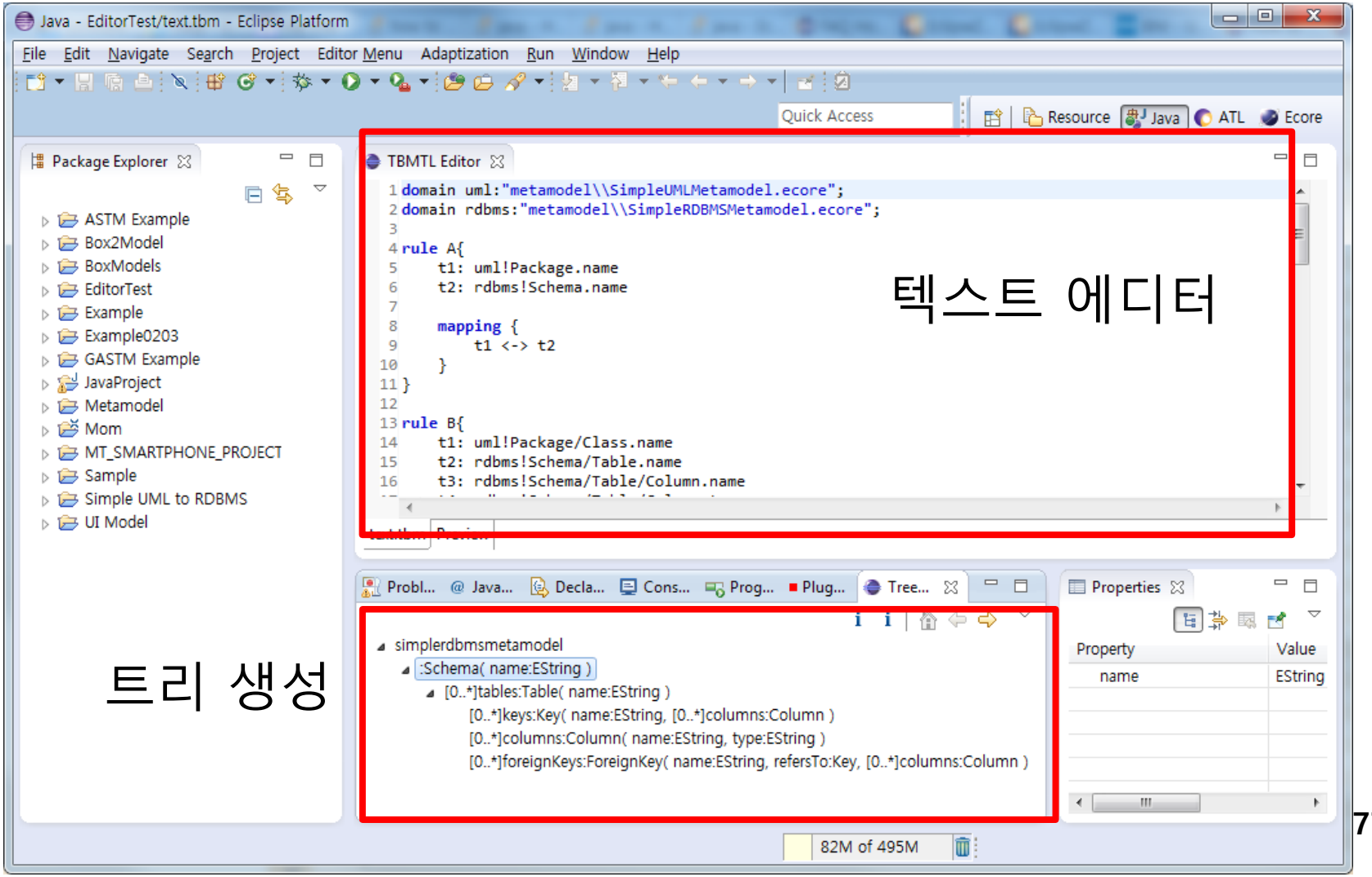


C/C++에서 Java로 변환 차이점

C/C++	Java	비고
virtual	abstract	virtual인 상태에서 메서드의 구현부가 없을 때
const	final	
~	close	자바에는 소멸자가 없음
int a[2]	int[] a= new int[2]	
external		자바는 external이 없음
long long	long	자바는 2개를 연속하는 타입을 가지지 않음
long double	double	
파일내의 메소드, 속성	클래스 내부로 이동	C++은 C 스타일을 혼용해서 사용할 수 있음
struct	class	자바는 struct 가 없음 class로 취급
다중상속	첫번째 상속 두번째는 구현으로 변경	C++은 인터페이스가 없으므로 구분이 되지 않음
include	import	
bool	boolean	
매크로		자바 제외
goto	continue	

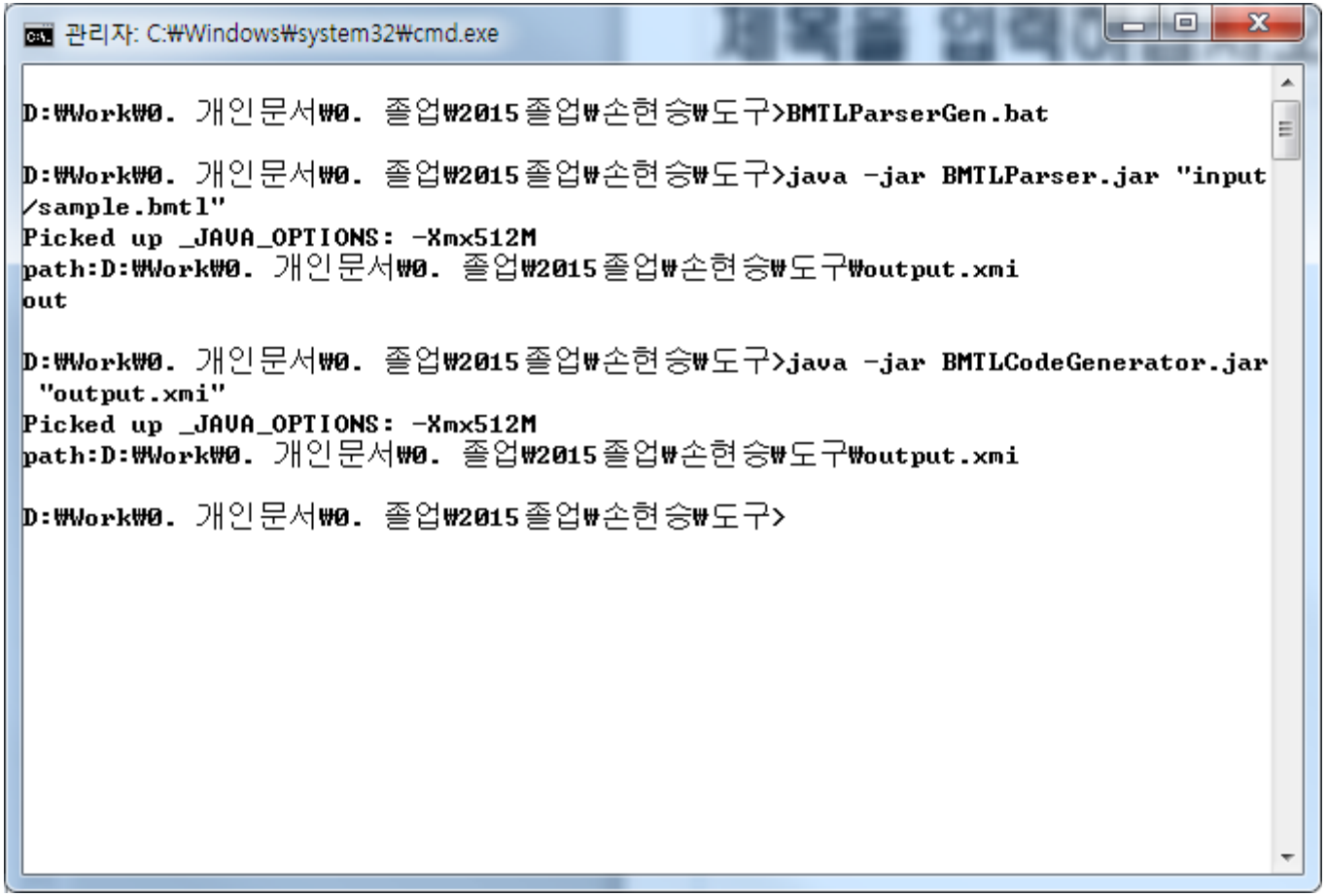
도구시현

➢ 모델 변환 도구(이클립스 플러그인)



도구시현

- 모델 변환 언어
  - ✓ sample.bmtl → BMTLParser.jar → output.xmi
  - ✓ output.xmi → BMTLCodeGenreator → Rule.java



도구시현

- ASTM 코드 생성
  - ✓ Figure.h → xCodeParser.jar → Figure.h.sastm
  - ✓ Figure.cpp → xCodeParser.jar → Figure.cpp.sastm
  - ✓ Figure.h.sastm xCodeParser.jar → Figure.h
  - ✓ Figure.cpp.sastm xCodeParser.jar → Figure.cpp
  - ✓ Figure.h.sastm xCodeParser.jar → Figure.java

```
path:D:₩work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구₩output₩Figure.cpp.sastm
Save complete!

D:₩Work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구>java -jar xCodeParser.jar -gcpp
 "output/Figure.h.sastm"
Picked up _JAVA_OPTIONS: -Xmx512M
path:D:₩Work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구₩output₩Figure.h.sastm
output/Figure.h

D:₩Work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구>java -jar xCodeParser.jar -gcpp
 "output/Figure.cpp.sastm"
Picked up _JAVA_OPTIONS: -Xmx512M
path:D:₩Work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구₩output₩Figure.cpp.sastm
output/Figure.cpp

D:₩Work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구>java -jar xCodeParser.jar -gjav
a "output/Figure.h.sastm"
Picked up _JAVA_OPTIONS: -Xmx512M
path:D:₩Work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구₩output₩Figure.h.sastm
path:D:₩Work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구₩output₩Figure.cpp.sastm
output/Figure.java

D:₩Work₩0. 개인문서₩0. 졸업₩2015졸업₩손현승₩도구>
```

6. 결론 및 향후 연구

- 기존 순공학(MDA) 또는 역공학(ADM) 방법을 병합함
- 기존 AST기법은 언어 종속 문제를 ASTM으로 해결
- 소프트웨어 품질관리를 위한 이종 코드 변환 프레임워크 제안
  - ✓ 개발 프로세스 정의
  - ✓ 양방향 모델 변환 방법 제시
  - ✓ 양방향 모델 변환 언어와 엔진 제안
  - ✓ 코드에서 ASTM으로 변환 방법 제안
  - ✓ ASTM을 코드로 변환 방법 제안
  - ✓ 모델 변형 엔진을 새로 개발

- 모델에서부터 코드 생성과 이종의 코드 생성 가능

- 향후 연구
  - ✓ ASTM 수정 보완(없는 엘리먼트 추가)
  - ✓ 정규화 된 방법 통한 생성된 모델 검증
  - ✓ 시퀀스, 스테이트 다이어그램 확장을 통한 완성도 향상

The End

Thank You !