A Method of Handling Metamodel based on XML Database for SW Visualization

Tuesday, February 16, 2016

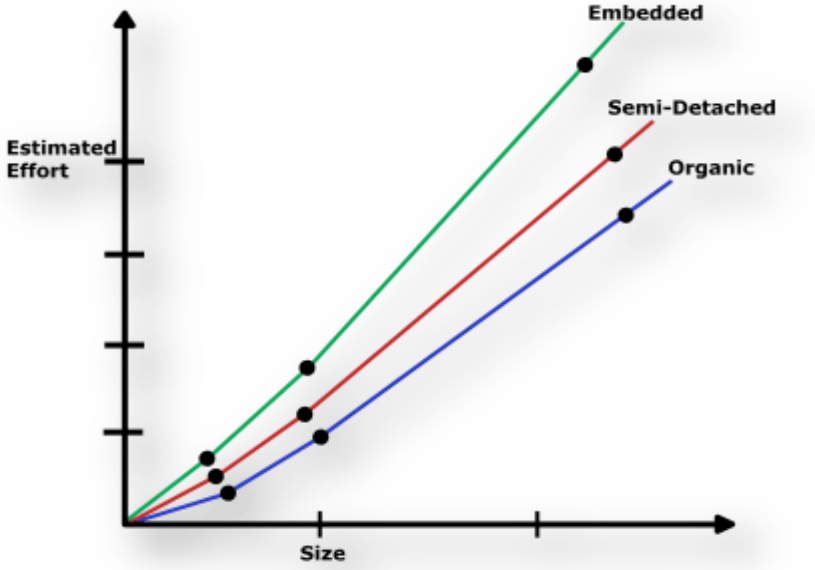
Hyun Seung Son
Software Engineering Lab.
Hongik University

Contents

1. Motivation
2. Related Works (ASTM, XML databases)
3. Our metamodel method
4. Case Study
5. Conclusions

1. Motivation

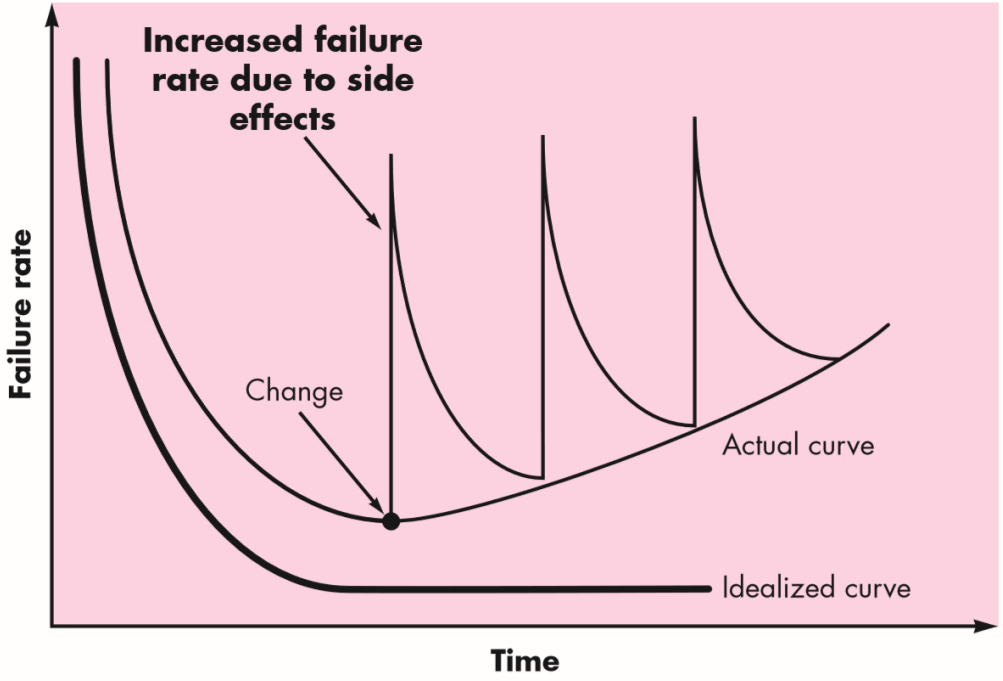
- The property of software
  - Has Invisibility, Complexity, and Changeability
  - Also, depends on each individual maturity level of developer
- Therefore, software is difficult to develop high quality software product
  - Which spends much more cost and time consuming with huge scale project



COCOMO Model
Effort Vs. Product size

1. Motivation cont.

- About Changeability of software :
  Due to being continuously modified and changed
  - We spend more than 60 percent of the whole development costs for just maintenance
  - Increase the error according to changing code on the maintenance : Bathtub Curve

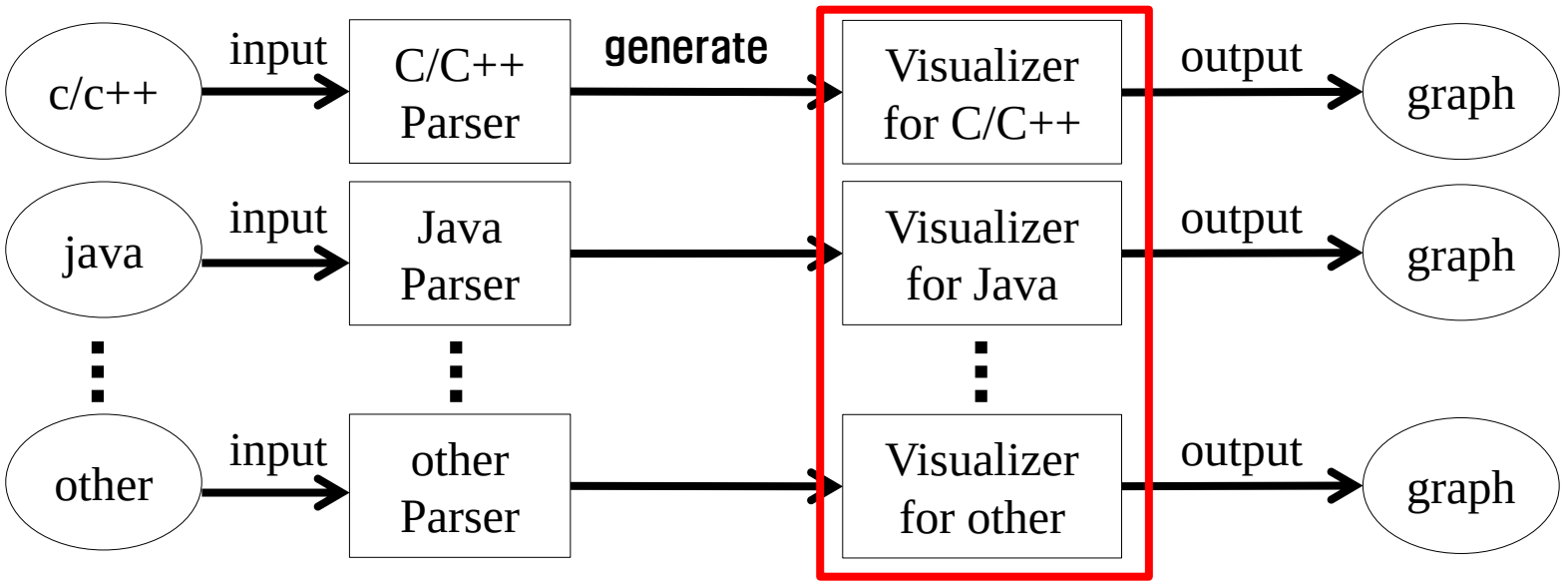


1. Motivation cont.

- Now, the software needs to manage the quality because of
  - increasing the software complexity,
  - frequently changing the user requirements
- How to guarantee high quality of software
  - Assess with Certification Models such as GS, CMMI, SPICE, TMMi,
  - Software Methodology, Process, Tools
  - Test Process such as TPI
  - White/Black Box Testing: impossible to do complete testing
- Therefore, How could you develop quality of software?
  - Our Issue: each SW developer has a Bad Habit to write programming !!!
  - We try to show the internal structure of the code for the SW visualization

Our previous research

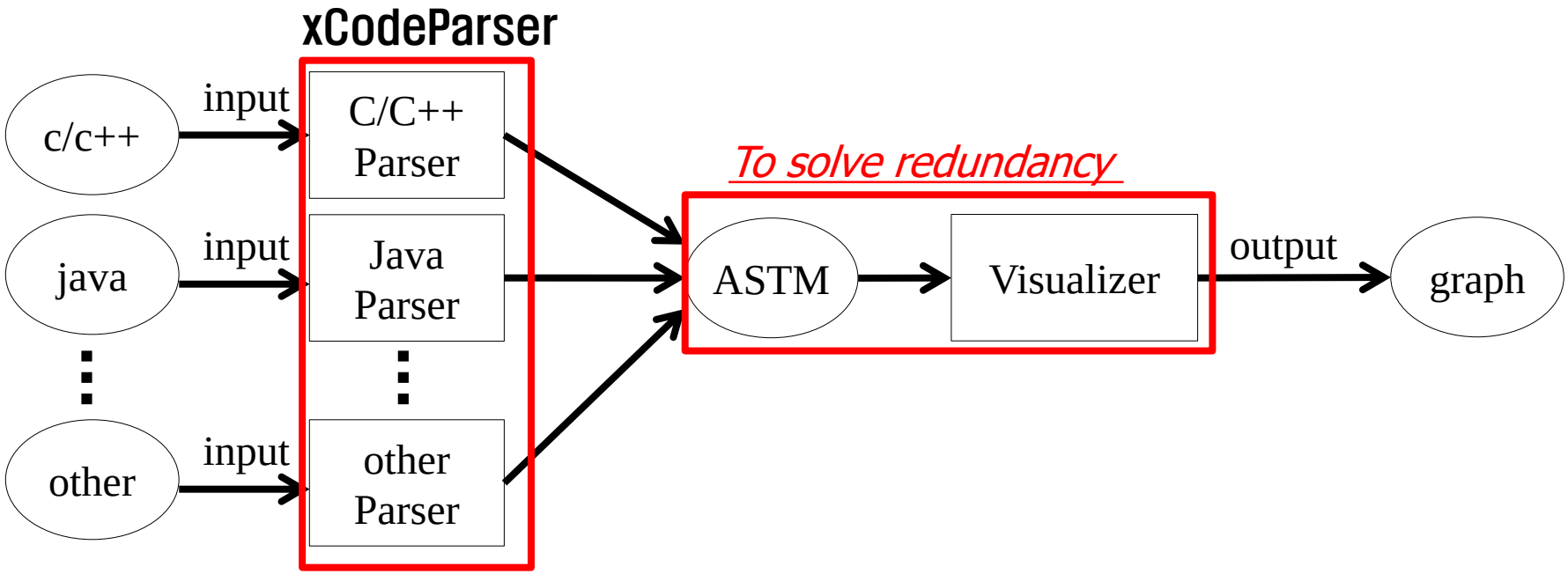
- The existing SW Visualization methods have problems :
  - For the SW visualization, it need many tools such as *Source Navigator*, *Graphviz*, and a parser, so on.
- Generally, the parser generates *Abstract Syntax Tree* (AST) during compiling the program code
  - The existing ASTs are not compatible with other ASTs due on the specific parser (redundancy)



Our previous research cont.

❑ Why do we need an ASTM Standard?

- The existing parser need many AST generators to possibly visualize the ASTs
- Do not interoperability between one AST and other ASTs



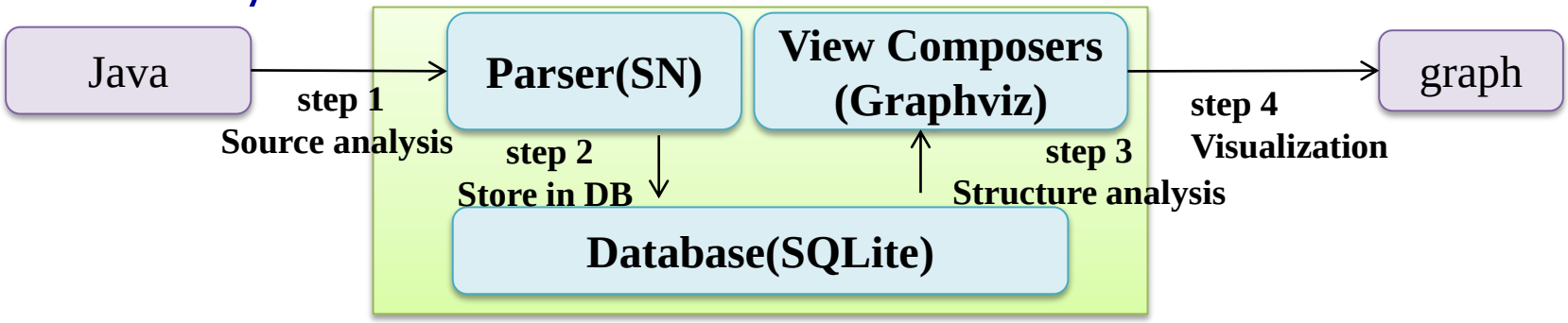
Our previous research cont.

- For SW visualization, it is required that
  1. The parser generates the abstract syntax tree
     - The parser generates ASTM from a program code such as C, C++, or Java.
  2. The XML database need to save ASTM data
     - The XML databases save the ASTM data
  3. The visualizer needs to generate a graph
     - The visualizer generates the graph from the ASTM

- Through this process, we can reverse the architecture from the program code

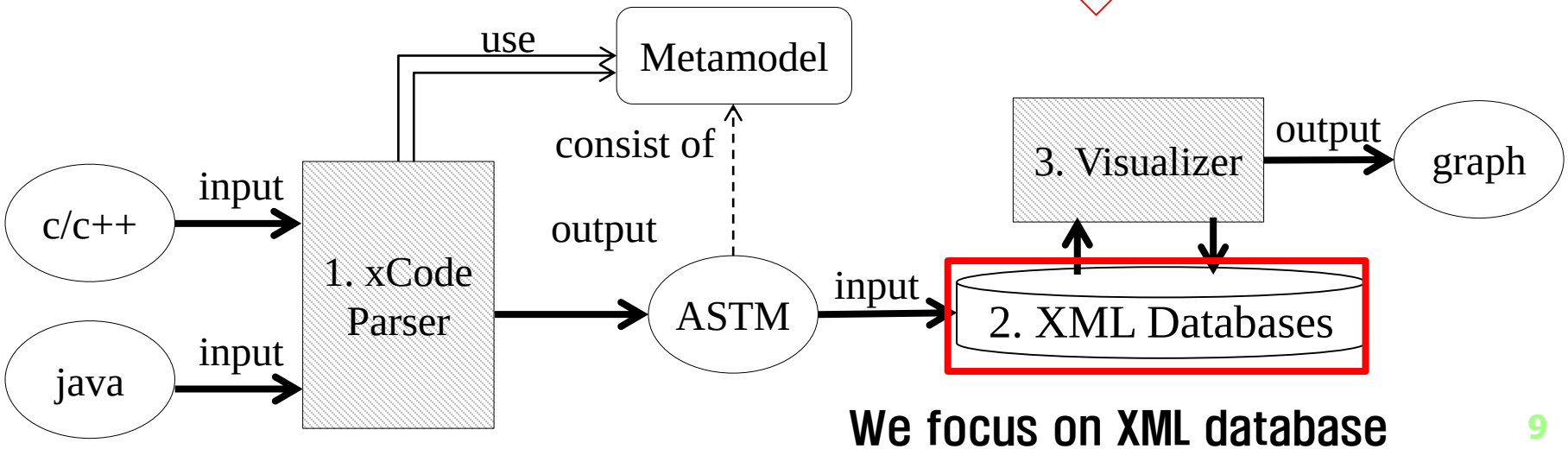
A whole structure for SW visualization

- The previous approach: used the open source tool chains on Code Static analysis



- This is current approach: used the ASTM

changing



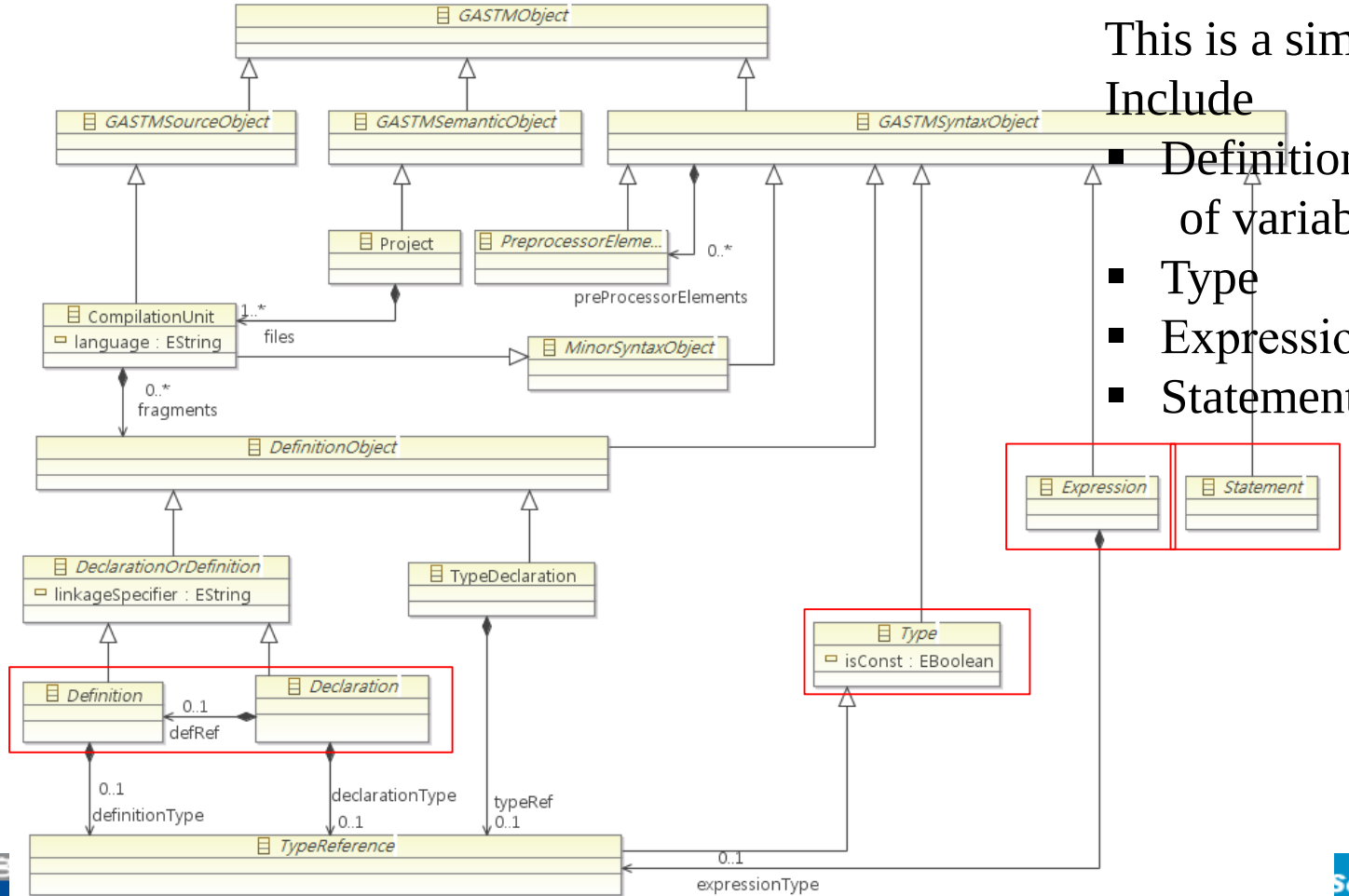
We focus on XML database

What is Abstract Syntax Tree Metamodel (ASTM) ?

- *Abstract Syntax Tree Metamodel (ASTM)* - OMG's standard
  - which is metamodel of abstract syntax tree with the existing compilers
  - represent the structure of source code with Abstract Syntax Trees
- The main purpose of the ASTM
  - Easily interoperate the metadata repository between program codes such as software modernization, platforms, and heterogeneous environment
- The ASTM has
  - 193 elements to represent the full AST from the existing programming languages such as *C, C++, C#, Java, Ada, VB/.Net, COBOL, FORTRAN, Jovial,* and so on

Abstract Syntax Tree Metamodel (ASTM)

- GASTMObject (that is, a Root: the general ASTM)
  - Consists of SourceObject, SementicOjbect, and SyntaxObject.
  - ASTM consists of total 193 of metaclasses from three objects.



This is a simple ASTM Structure
Include

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

A whole Abstract Syntax Tree Metamodel : implement 193 metaclasses based on OMG standard

- 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

Abstract Syntax Tree Metamodel (ASTM) cont.

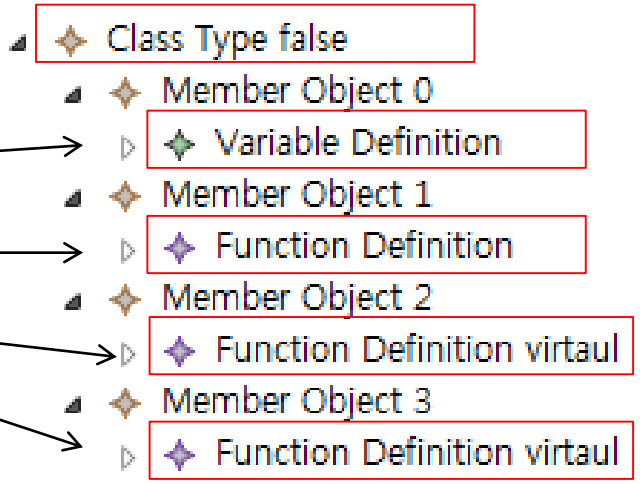
- ByValueFormalParameterType -> FormalParameterType
- ByReferenceFormalParameterType -> FormalParameterType
- Public -> AccessKind
- Protected -> AccessKind
- Private -> AccessKind
- UnnamedTypeReference -> TypeReference
- NamedTypeReference -> TypeReference
- ExpressionStatement -> Statement
- JumpStatement -> 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

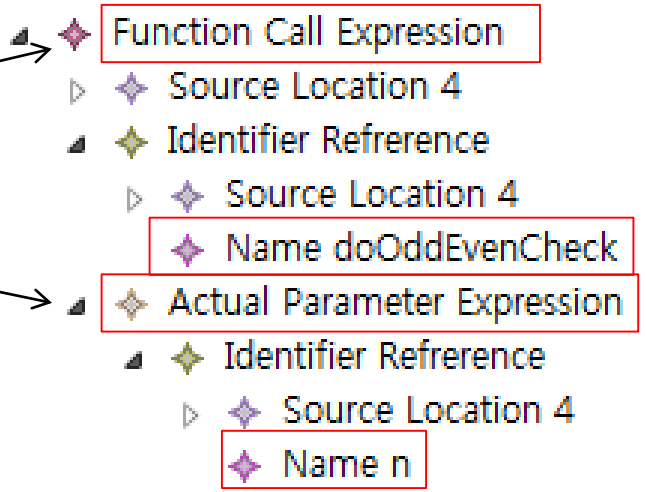
- Add -> BinaryOperator
- Subtract -> BinaryOperator
- Multiply -> BinaryOperator
- Divide -> BinaryOperator
- Modulus -> BinaryOperator
- Exponent -> BinaryOperator
- And -> BinaryOperator
- Or -> BinaryOperator
- Equal -> BinaryOperator
- NotEqual -> BinaryOperator
- Greater -> BinaryOperator
- NotGreater -> 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

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;
}
```



What is XML database ?

- XML database ?
  - A database system for storing the data in XML format directly
- Two ways to save XML data :

1. RDBMS to store the XML
   - RDBMS has various product such as IBM DB2, Microsoft SQL Server, Oracle and PostgreSQL
   - It has advantageous to store the data of XML with the existing database
   - But needs to use the existing SQL statements and XQuery to query XML data at the same time, which will not be saved in a specific format
2. Native XML databases
   - The native XML databases extract the data through XQuery or XPath using only data of XML
   - Which have various types such as BaseX, Qizx, eXist or MarkLogic Server

- In this paper, we choose the native XML databases to save XML data
  - Problem : we must write twice a query on RDBMS
  - But native XML databases is available with a single query

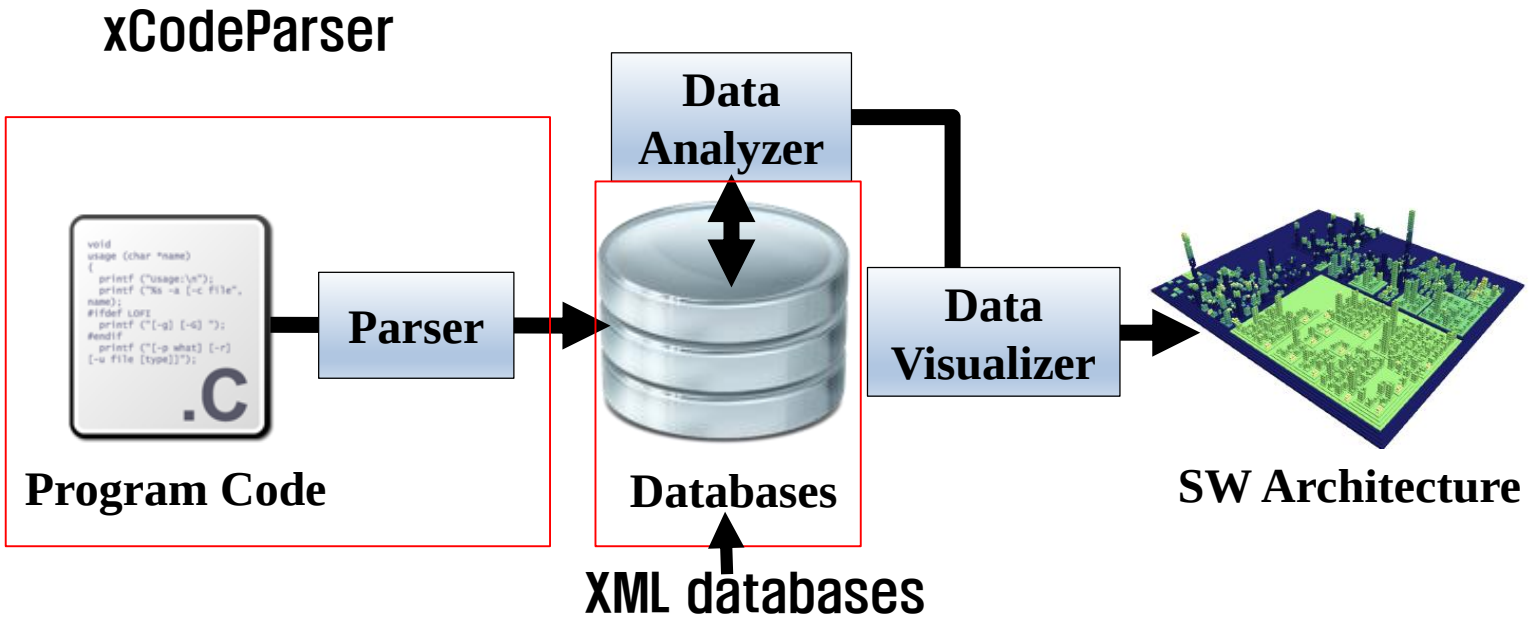
The survey of native XML databases

- The specific features of native XML database is shown in table

Property \ Name	BaseX	Qizx	eXist	MarkLogic Server
Native Language	JAVA	JAVA	JAVA	C++
XQuery 3.0	Supported	Supported	Partial	Partial
XQuery Update	Available	Available	Commercial	Commercial
XQuery Full Text	Supported	Supported	Commercial	Commercial
EXpath Extension	Available	No	No	No
EXqueryExtension	Available	No	Available	No
XSLT 2.0	Available	Available	Available	Available

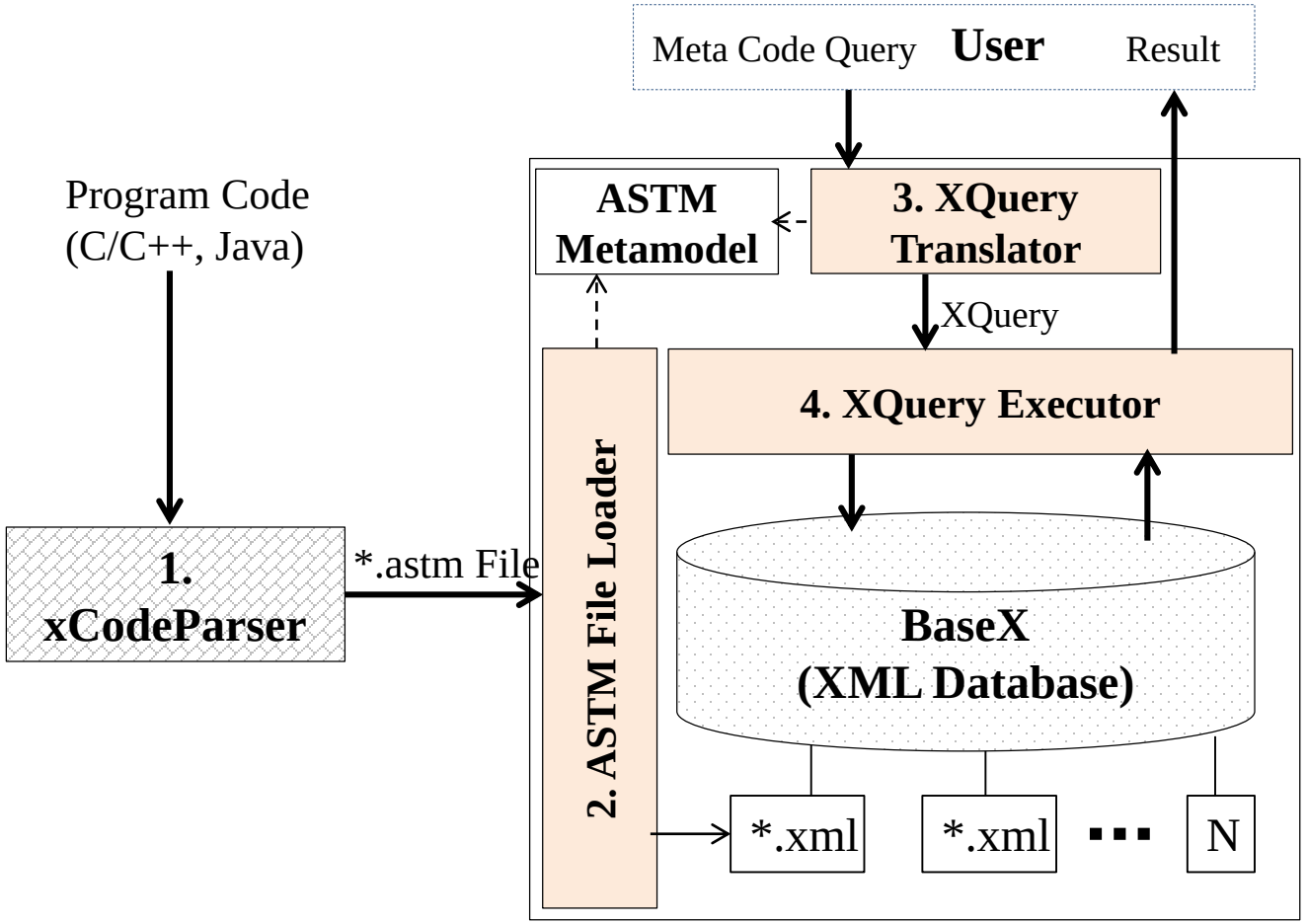
3. Our metamodel method

- To implement the software visualization, the developer is necessary to process three steps:
  1) The parser to generate the Abstract Syntax Tree (AST) from program code
  2) The data analyzer to store data using databases, and to extract the related data of the source code from AST
  3) The data visualizer to represent various graphs from the analyzed data



Proposed structure of our metamodel method

- We propose the metamodel based integratation data repository to save the ASTM files, and to use the data analyzer in the repository

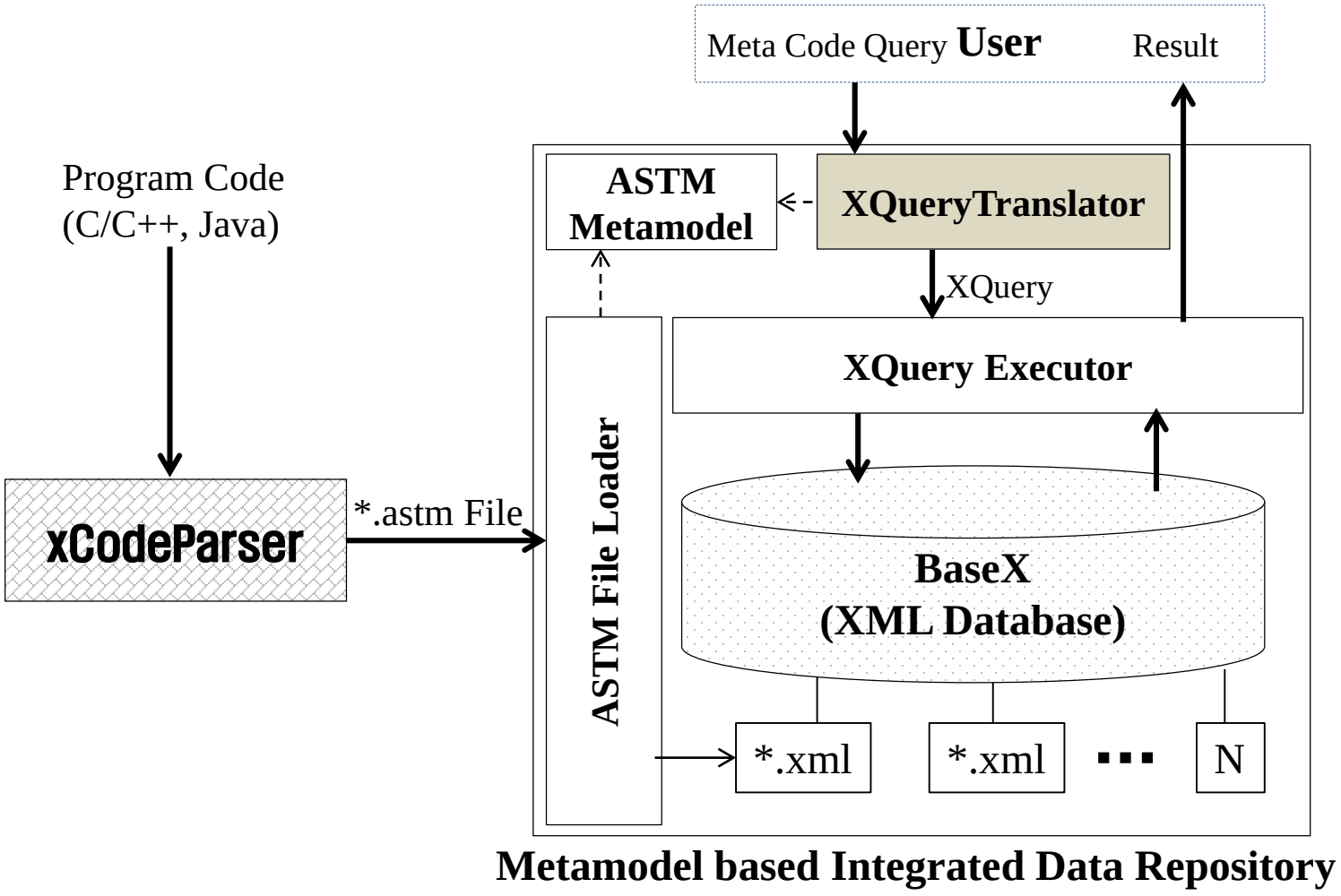


XML DB Process

1. xCodeParser: generate ASTM file from code
2. ASTM File Loader: save ASTM in XML database
3. XQuery Translator: translate XQuery from proposed query language
4. XQuery Executor: run translated XQuery using BaseX API

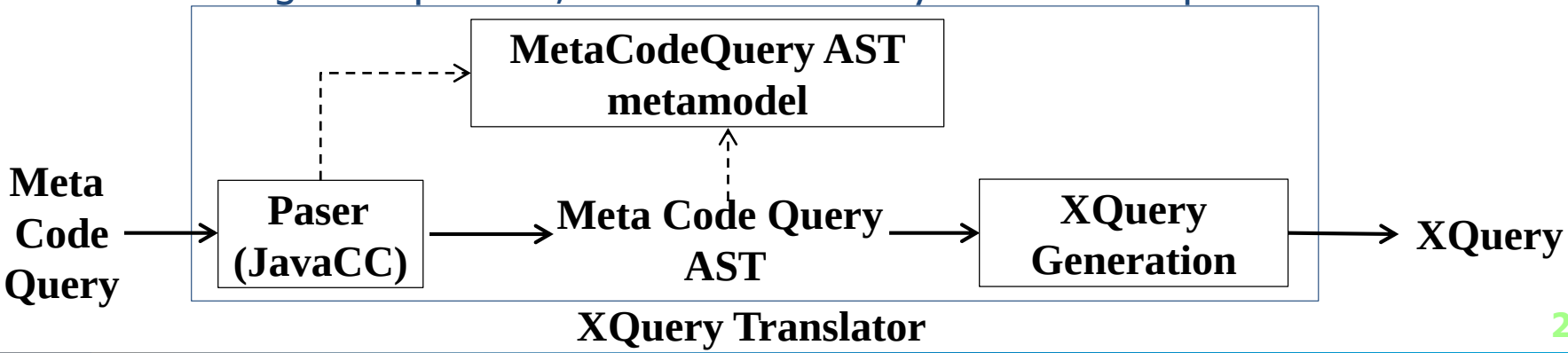
Metamodel based Integrated Data Repository

Proposed structure of our metamodel method



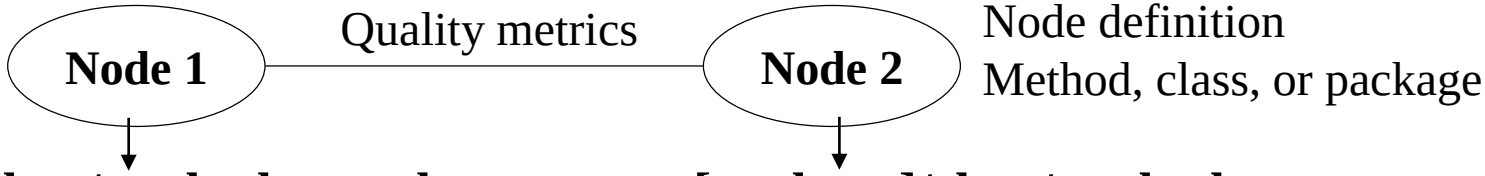
Proposed structure of our XQuery Translator

- The XML database(BaseX) perform a search using XQuery
  - It is difficult to find the metamodel because the XQuery searches the only data of XML.
- Thus, we propose a new query language named Meta Code Query
  - That allows the user to easily view the data in the code
  - The XQuery Translator converts to XQuery from them
  - It does easily view the program code information with a simple command line to define the pattern of the metamodel of the ASTM
  - Finally, the XQuery executor performs the converted query statement to XML databases, and transfers the results to the user
  - Through this process, the user can easily show the required data



Our Meta Code Query Language

- The proposed Meta Code Query Language
  - In the code of visualization, the two nodes is given by a line
  - The line indicates the quality metrics between the nodes as shown figure
  - In figure, node is a name of package, class or method
  - The quality metrics is code pattern

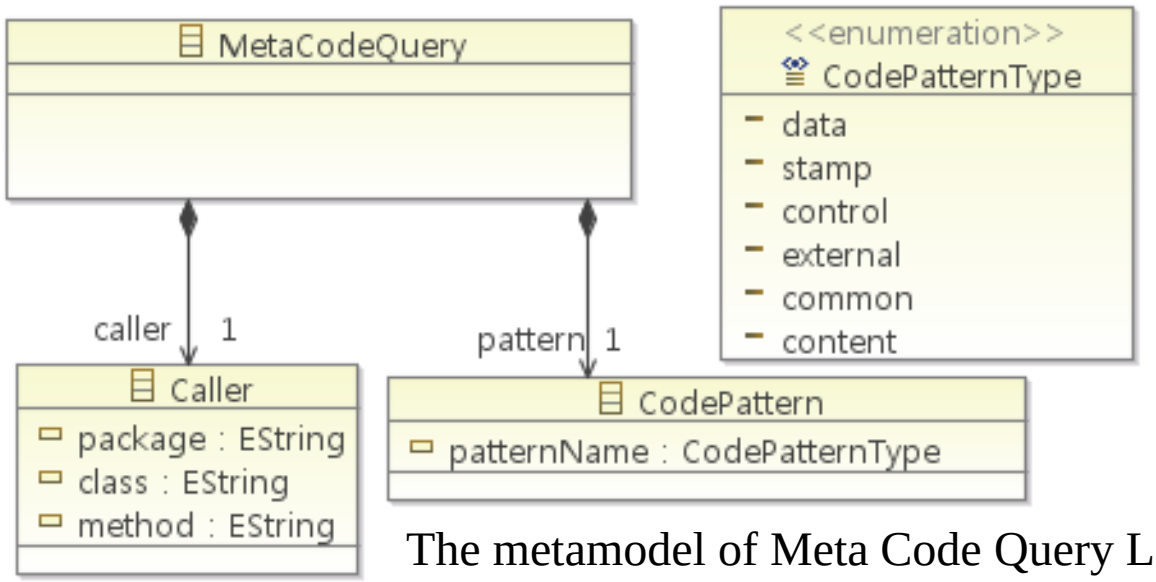


- In this aspect,
  - Starting node is the method, class or package, in accordance with the level of abstraction for searching data
  - If the class level, you will see all methods in that class.
- In conclusion, when a pattern matches the start node and code pattern, it can be obtained for the second node

Our Meta Code Query Language cont.

- The EBNF of Meta Code Query Language

```
<MetaCodeQuery> ::= [ <package> ] "/" <class> "/"
                    <method> "->" <code pattern>
<package> ::= <Identifier>
<class> ::= <Identifier>
<method> ::= <Identifier>
<code pattern> ::= "data" | "stamp" | "control" | "external" |
                   "common" | "content"
<Identifier> ::= ["a"-"z", "A"-"Z", "_"]
                 (["a"-"z", "A"-"Z", "_", "0"-"9"])*
```



The metamodel of Meta Code Query Language

Our Meta Code Query Language cont.

- A example of transformation of Meta Code Query

Meta Code Query : / **<class>** / **<mehtod>** -> control

XQuery :

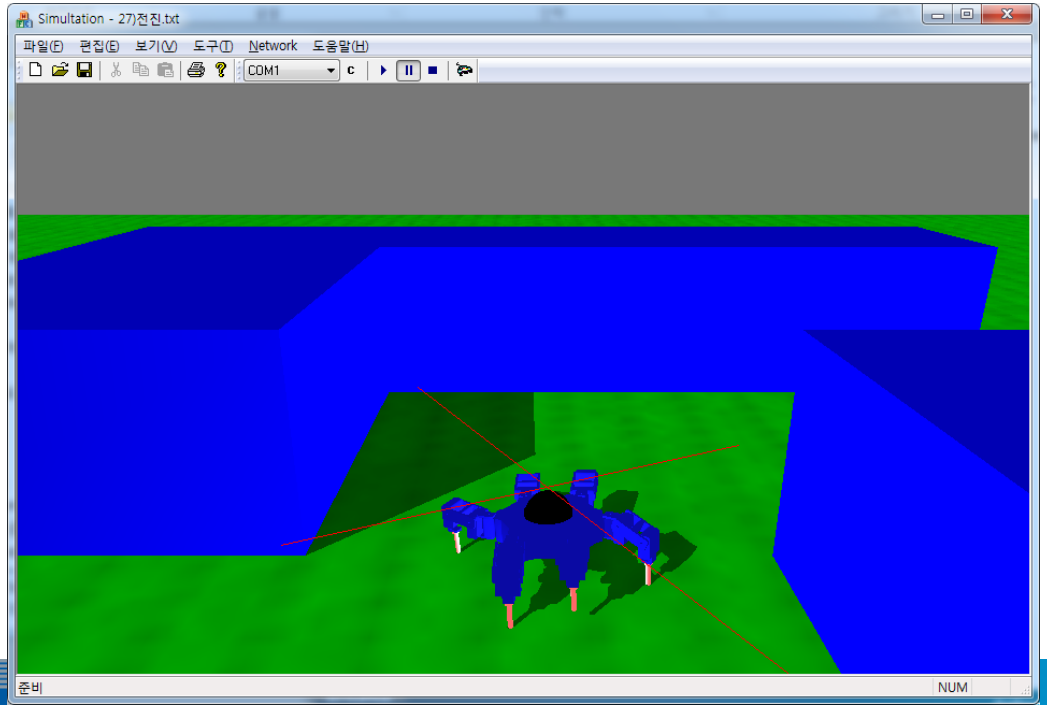
```
for $call in //project//fragments
 let $caller        := $call/identifierName/@nameString
 let $called        := $call/body//subStatements/condition/@xsi:type
 let $called_class  := $call/body//subStatements/condition/
                       calledFunction/qualifiers/identifierName/@nameString
 let $called_method := $call/body//subStatements/condition/
                       calledFunction/member/identifierName/@nameString
where $called = 'gastm:FunctionCallExpression'
      and contains(data($caller),'<class>::<method>')
```

Result:

```
<link codePattern=\"control\">
       <caller class = '{tokenize($caller,'::')[1]}'
                       method = '{tokenize($caller,'::')[2]}'></caller>
       <called class = '{$called_class}'
                       method = '{$called_method}'></called></link>
```

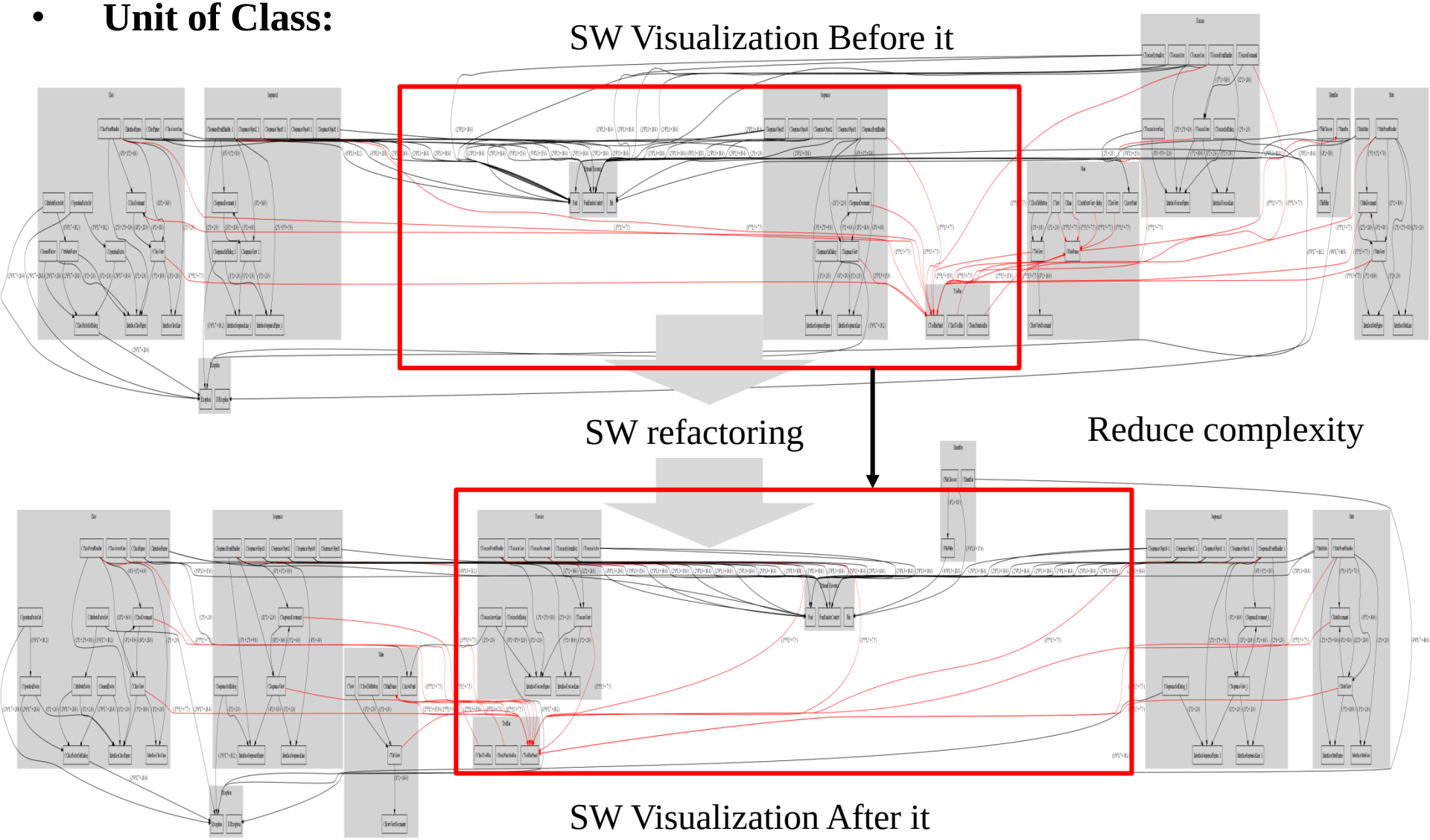
4. Case Study

- Target software : Multiple-joint Robot Simulator
  - This project was supported by the Human Resource Training Program for Regional Innovation and Creativity
  - A development tools of simulation control program for the multiple-joint robot (Korea Patent No. 10-0956839)
  - Program Language : C++
  - External Library : ODE(Open Dynamic Engine), MFC, Direct X



(common coupling) before/after Refactoring

- **Unit of Class:**



5. Conclusions

- The software visualization is
  - A method that extracts an architecture from program codes
  - The SWV organized the open source based tool-chain using Source Navigator, Graphviz, SQLite, Jenkins, etc. but have to need many tools
- The *Abstract Syntax Tree Metamodel* (ASTM) is
  - Useful to convert from the diverse program codes to *Abstract Syntax Tree*
  - Good for interoperability
- This paper
  - Develops the xCodeParser of ASTM based on OMG standard
  - Suggests the metamodel based on XML databases in a whole procedure for SW visualization
  - For handling metamodel, we proposed Meta Code Query Language and implement XQuery Translator
- Through visualizing software
  - Help to improve quality, and to reduce reuse & maintenance cost
  - In further research, we will develop visualizer to represent the various graph