Re-engineering Eclipse MDT/OCL for Xtext

Edward D Willink
MDT/OCL Project Lead
Eclipse Modeling Project

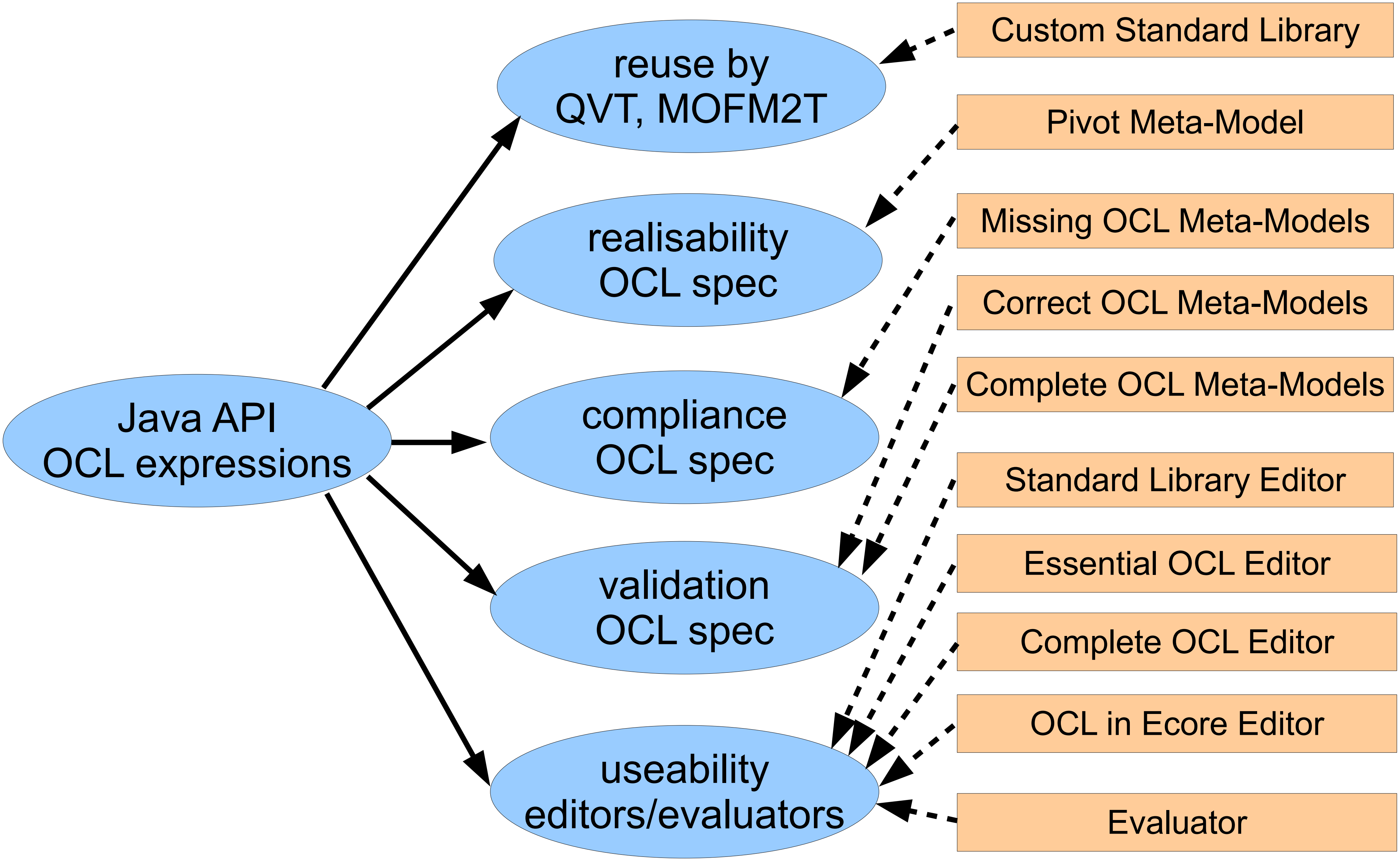
MODELS 2010
Workshop on OCL and Textual Modeling

3rd October 2010

Overview

- Eclipse MDT/OCL evolution to use Xtext
- Xtext impact
- Xtext/LPG performance comparison
- Xtext-mandated changes of approach
- Xtext-motivated revisions

MDT/OCL Evolution

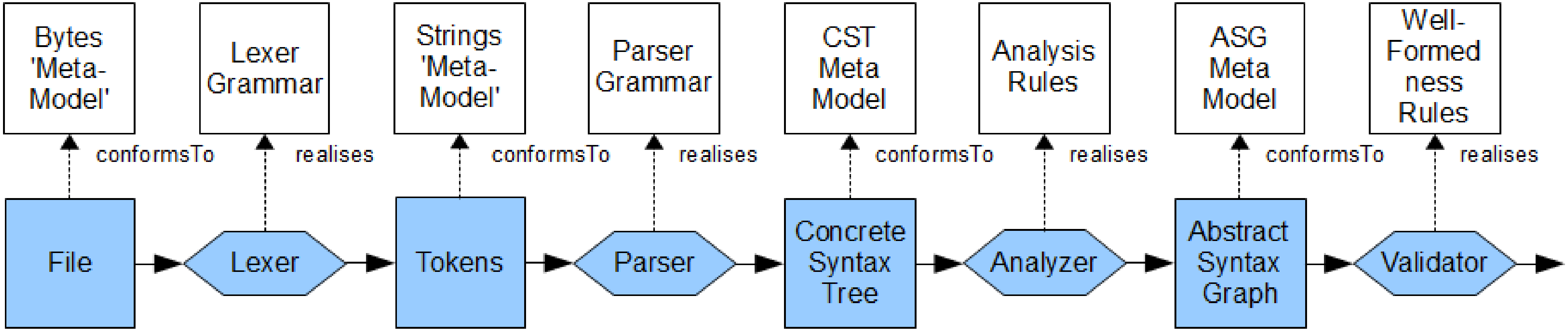


Adding Editors to MDT/OCL

- First attempt used basic SWT framework
- Second attempt used IMP
  - re-uses existing LPG LALR parsers
  - inherited realisation of editing idioms (highlighting)
- Third attempt uses Xtext
  - requires migration to LL parser (ANTLR)
  - inherited modeled generation of editing idioms
  - modeled generation of syntactical constructs

- Contrast old LPG+IMP with new Xtext approach

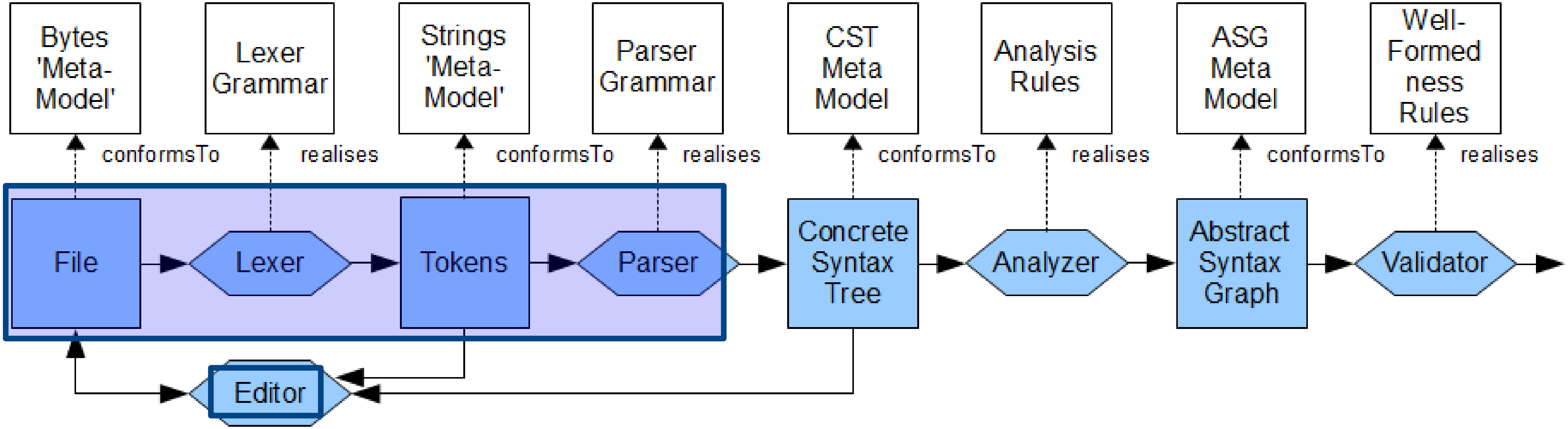
Basic Parser Architecture



- Lexer: Text -> Tokens
- Parser: Tokens -> CST Nodes
- Analyzer: CST Nodes -> ASG Nodes
- Validator: ASG Nodes -> Diagnostics

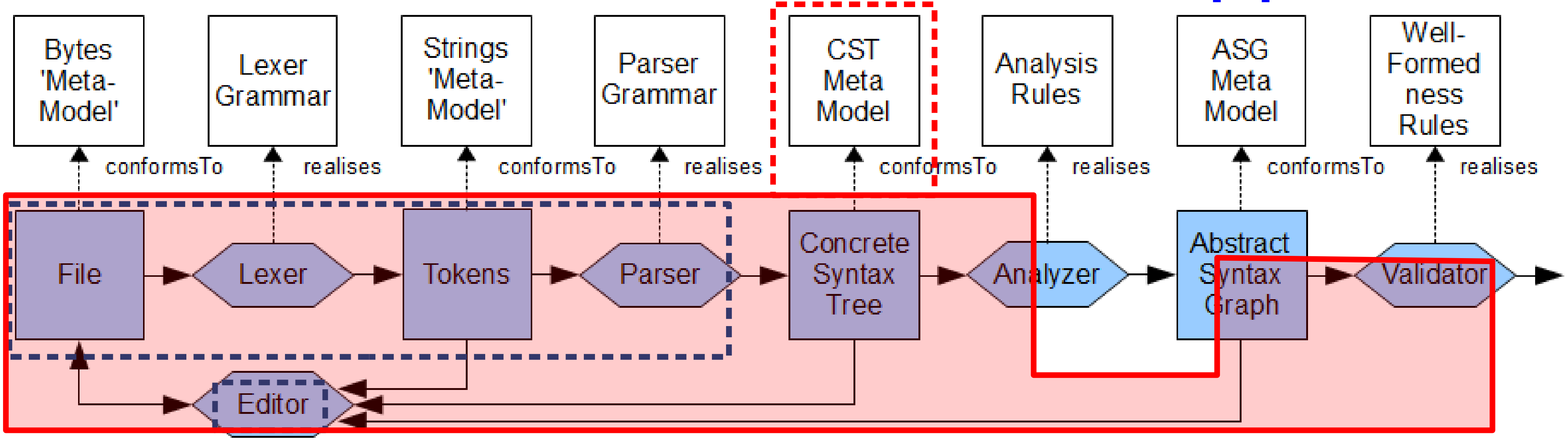
[OCL is complex: CST very different to ASG]

LPG Parser with IMP Editor



- Manual provision of
  - 2 lexer grammars and 1 parser grammar
  - parser action code to populate CST
- Auto-generation of
  - lexer, LALR parser

Xtext Parser and Editor Support



- Manual provision of
  - combined parser grammar
  - optional CST meta-model
- Auto-generation of
  - lexer, LL parser, and analyzer framework
  - editor with rich model-driven features
- Validation using CHECKS language or Java

LPG Action Code

- CollectionRange: 1..10

```
CollectionRangeCS ::= OclExpressionCS '..' OclExpressionCS
  /.$BeginCode
    CollectionRangeCS result = CSTFactory.eINSTANCE.createCollectionRangeCS();
    result.setExpressionCS((OCLExpressionCS)getRhsSym(1));
    result.setLastExpressionCS((OCLExpressionCS)getRhsSym(3));
    setOffsets(result, (CSTNode)getRhsSym(1), (CSTNode)getRhsSym(3));
    setResult(result);
    $EndCode
  ./
```

- CST access woven into code
  - not checked till Java compiled/run
- Significant and repetitive actions
  - fragile policies, casts, magic numbers

Xtext 'Action Code'

CollectionRange: 1..10

```
CollectionRangeCS ::= expressionCS=OclExpressionCS '..'
                      lastExpressionCS=OclExpressionCS
```

- CST woven into grammar
  - declarative: checkable / generateable
- No code
- In practice, two productions can be merged

```
CollectionLiteralPartCS ::= expressionCS=OclExpressionCS
                 ('..' lastExpressionCS=OclExpressionCS)?
```

Cross-references

PathName **A::B::c**

- LPG - cross-reference is an unresolved String

```
pathNameCS ::= Identifier
/* Code not shown */
pathNameCS ::= pathNameCS '::' Identifier
/* Code not shown */
```

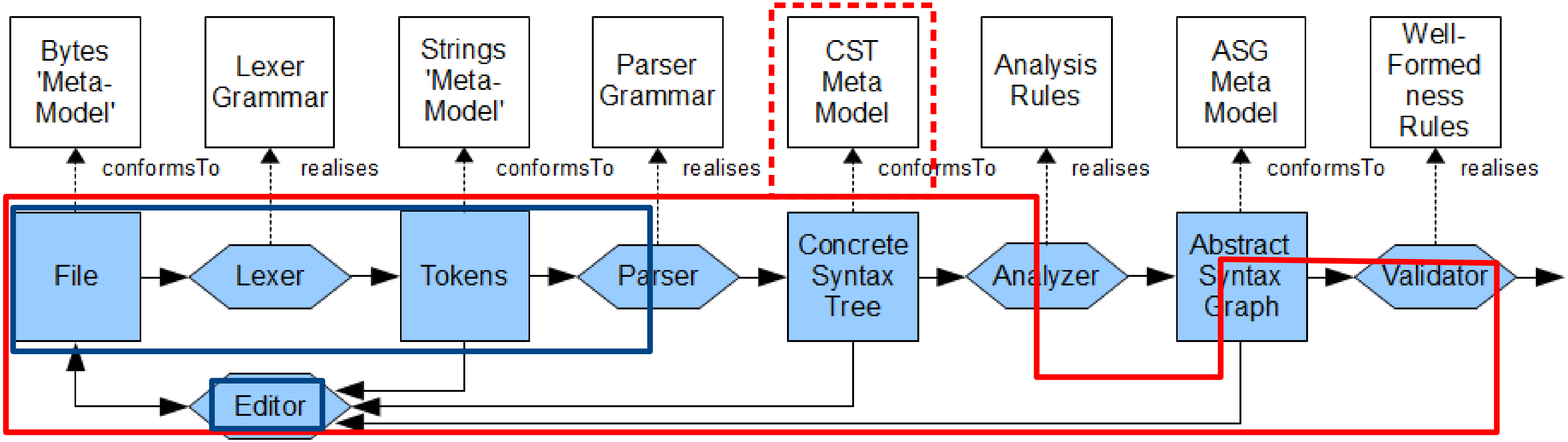
  - left recursion

- Xtext - cross-reference is a resolved EObject

```
pathNameCS returns PathNameExpCS:
   (namespace+=[Namespace|Identifier] '::')*
  element=[NamedElement|Identifier];
```

  - No code, CST Declarations woven into grammar
  - EObject Reference Declarations woven into grammar
  - =, += and ?= CS assignments
  - (), *, +, ? BNF operators

Cross-reference impact



- Cross-reference requires analysis

```
pathNameCS returns PathNameExpCS:
  (namespace+=[Namespace|Identifier] '::')*
  element=[NamedElement|Identifier];
```

  - to locate the `Namespace` for an `Identifier`

- Xtext provides a default scope resolution
- OCL defines explicit Environment lookup

Performance : Grammar Size

- Simple examples
  - Xtext line count is three times smaller
- MDT/OCL 3.0.0 implementation comparisons

Line counts	LPG 2.0.17	Xtext 1.0.0
Lexer grammars	251	395
Parser grammar	1485	
Templates	1000	library
Java support	1040+library	library
Total	~2800	~400

  - Similar real application, similar editorial style
  - Xtext at least 5 times smaller
    - and it autogenerates an editor too

Performance : Parser Size

- MDT/OCL 3.0.0 implementation comparisons

class file sizes	LPG 2.0.17	Xtext 1.0.0
Lexers and Parsers	221 kB	2370 kB
Semantic analysis	excluded	excluded
Total	~220 kB	~2400 kB

- Similar grammar
- Xtext about 10 times larger
  - different grammar generated for editor
    - extra completion assist functionality
  - another 1MB

Performance : Speed

- 350 line Complete OCL example (Royal & Loyal)
- MDT/OCL 3.0.0 implementation comparisons

Time in milliseconds	LPG 2.0.17	Xtext 1.0.0
First parse	1800	4800
Files read for first parse	2	6
Average of 100 reparses	**97**	**1114**
Files read for reparse	1	1

- Similar real application, same example
- Xtext about 11 times slower

Performance: Summary

- Xtext is 5 times smaller source size
  - fundamental technology advance
  - massive ergonomic gain
- Xtext is 10 times larger classes sizes
- Xtext is 11 times slower execution
  - Xtext 1.0.0 is not perfect
    - maybe better in Xtext 2.0
  - use of ANTLR and LL is not fundamental
    - maybe a conversion to LALR is appropriate
  - size, at least, very sensitive to grammar approach

Left Recursion

- LALR uses left recursion extensively

```
multiplicativeCS ::= unaryCS
multiplicativeCS ::= multiplicativeCS '*' unaryCS
...
multiplicativeCS ::= multiplicativeCS '/' unaryCS
...
```

- ANTLR can only right recurse, but in Xtext

```
multiplicativeCS: unaryCS (('*'|'/') unaryCS)*;
```

  - recursion replaced by repetition

Lookahead in OCL

`a->b(c,d` could be

`a->b(c,d|e`	an IteratorExpCS
`a->b(c,d,e`	an OperationCallExpCS

[iterator names (e.g. `b`) are not reserved/known]

- Difficult, inelegant to disambiguate with LALR(1)
  - compile time integrity check
- Must use backtracking in LL
  - Xtext hides any ANTLR checking
  - use an LALR copy for checking

Flexible operation parsing 1

Concrete Syntax parsed as a larger language

```
ID ('.'|'->') ID '('
  (expr (':' type)?
    (',' expr (':' expr)?)*
    (';' expr)*
    ('|' expr (',' expr)*)?
  )?
')'
```

- `a.b(c=5:d,e;f|g,h)` is parsed successfully
  - a semantic rather than syntactic error

Flexible operator parsing 2

OCL 2.0 and 2.2 (and QVT) change precedences

- different grammars
- one larger flexible grammar

```
infixOp ::= 'and'|'or'|'xor'|'implies'|'+'|'*'....
prefixOp ::= 'not'|'-'
expr ::= unary (infixOp unary)*
unary ::= (prefixOp)* atomic
```

all operators parsed with equal precedence

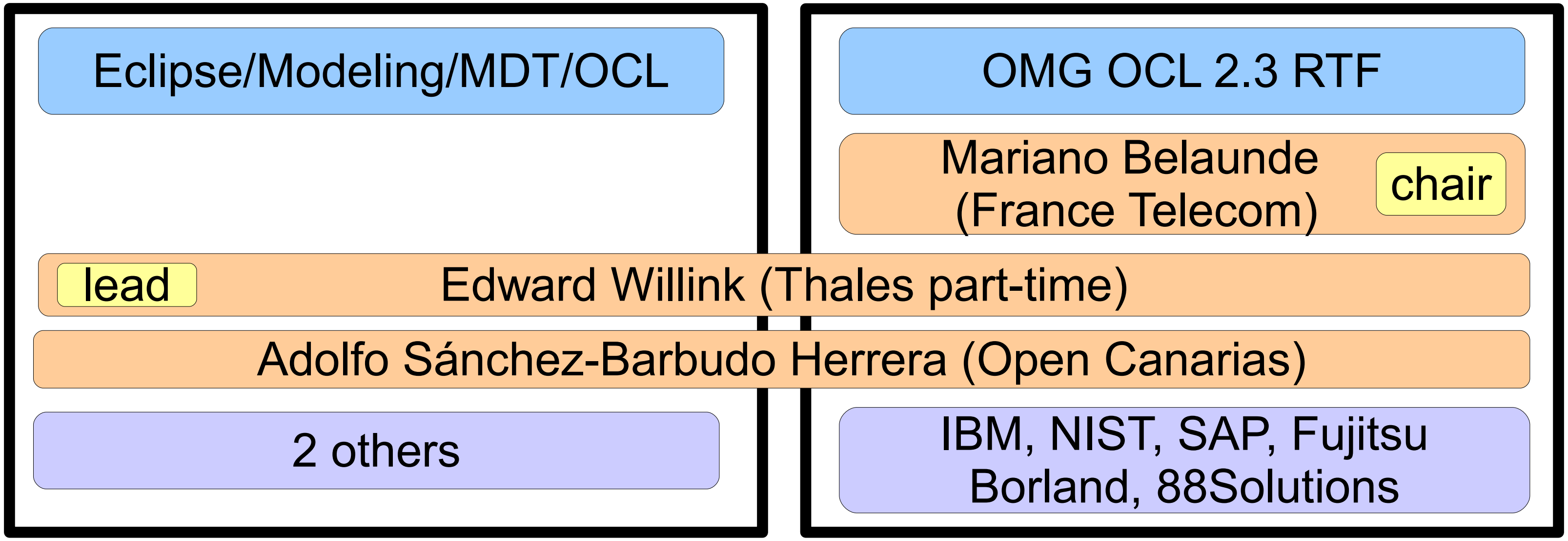
- 'standard' library defines precedence, associativity
- semantic analysis uses 'standard' library

- 5 fold reduction in ANTLR class sizes
  - solves 65536 byte class size limit

Summary

- Xtext facilitates an IDE for Eclipse MDT/OCL
- Xtext does many things very well
- Xtext cannot emulate all traditional approaches
  - Xtext seems to have a better way
- Xtext motivates a major rethink

Active OCL Participation



Eclipse/Modeling/MDT/OCL

OMG OCL 2.3 RTF

Mariano Belaunde
(France Telecom)

chair

lead

Edward Willink (Thales part-time)

Adolfo Sánchez-Barbudo Herrera (Open Canarias)

2 others

IBM, NIST, SAP, Fujitsu
Borland, 88Solutions

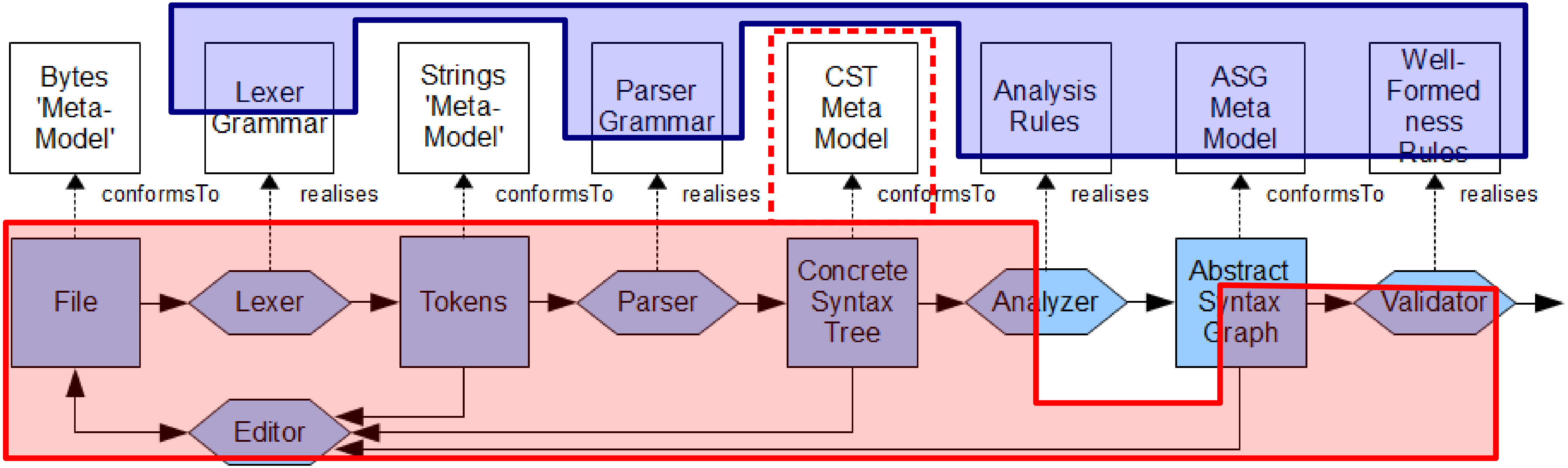
Open Source
Individual participation

Open Specification
Corporate participation

Original code from IBM
maintained by IBM until 2009
(Christian Damus)

Intellectual Property checks

OCL Specification-driven



- Make OCL specification consumable
  - useable complete lexer/parser grammar
  - useable/accurate analysis/validation constraints
- Model-driven analyzer framework
- OCL-driven validation

OCL-driven validation

- Eclipse Helios adds Validation Delegate support
  - OCL can be embedded in Ecore annotations
  - OCL is then executed during model validation
    - this works in Xtext
- Add a validation reason to Complete OCL

```
context MyClass
  inv UpperCaseName
       ('\'' + name + '\' is not uppercase in ' + toDiagnostic())
                   : name.toUpperCase() = name
```

- Add OCL to Java code generation
  - avoid performance penalties