Translating OCL to Natural Language

David Burke and Kristofer Johannisson

Background

- Existing system for linking formal software specifications in OCL to Natural Language
- Based on the Grammatical Framework [Ranta]
- Can we make it scale to handle a case study?
  - Translating formal (OCL) specifications of the Java Card API into English

Motivation

The KeY Project: Integrate formal software specification and verification into the industrial software engineering process.

Observation:

- Formal specifications necessary for proving that a program is correct
- Informal specifications required by customers, managers, software engineers

Goals

- Link formal and informal specifications:
  - authoring and maintaining formal/informal specifications
  - presenting specifications to different audiences
- **Batch translation of existing formal specifications into natural language**

Case Study

Can we automatically translate an existing collection of non-trivial formal specifications into natural language of acceptable quality?

- OCL specifications of the Java Card API [Larsson, Mostowski]

```
context OwnerPIN::check(
    pin : Sequence(Integer),
    offset : Integer,
    length : Integer) : Boolean
...
post: (self.tryCounter > 0 and pin <> null and
         offset >= 0 and length >= 0 and
         offset+length <= pin->size()
         and Util.arrayCompare(self.pin, 0, pin,
                   offset, length) = 0
      ) implies
      (result = true and
         self.isValidated() and
         tryCounter = maxTries)
```

for the operation check (pin : Seq(Integer) , offset : Integer , length : Integer) : Boolean of the class javacard::framework::OwnerPIN the following holds : the following postconditions should hold : ... (*) if the tryCounter of the ownerPIN is greater than 0 and pin is not equal to null and offset is at least 0 and length is at least 0 and offset plus length is at most the size of pin and the query arrayCompare (the pin of the ownerPIN , 0 , pin , offset , length) to Util is equal to 0 , the result is equal to true and the query isValidated ( ) holds for the ownerPIN and the tryCounter of the ownerPIN is equal to the maxTries of the ownerPIN for the operation check (pin : Seq(Integer) , offset : Integer , length : Integer) : Boolean of the class javacard::framework::OwnerPIN the following holds : the following postconditions should hold : ... (*) if the **tryCounter** of the **ownerPIN** is greater than 0 and pin is not equal to null and offset is at least 0 and length is at least 0 and offset plus length is at most the size of pin and the query **arrayCompare** (the pin of the **ownerPIN** , 0 , pin , offset , length) to Util is equal to 0 , the result is equal to true and the query **isValidated** ( ) holds for the **ownerPIN** and the **tryCounter** of the **ownerPIN** is equal to the **maxTries** of the **ownerPIN** for the operation check (pin : Seq(Integer) , offset : Integer , length : Integer) : Boolean of the class javacard::framework::OwnerPIN the following holds : the following postconditions should hold : ... (*) if the tryCounter **of the ownerPIN** is greater than 0 and pin is not equal to null and offset is at least 0 and length is at least 0 and offset plus length is at most the size of pin and the query arrayCompare (the pin **of the ownerPIN** , 0 , pin , offset , length) to Util is equal to 0 , the result is equal to true and the query isValidated ( ) holds for the ownerPIN and the tryCounter **of the ownerPIN** is equal to the maxTries **of the ownerPIN**

For the operation **check (pin: Sequence(Integer), offset: Integer, length: Integer): Boolean** of the class **javacard::framework::OwnerPIN** , the following post-conditions should hold:

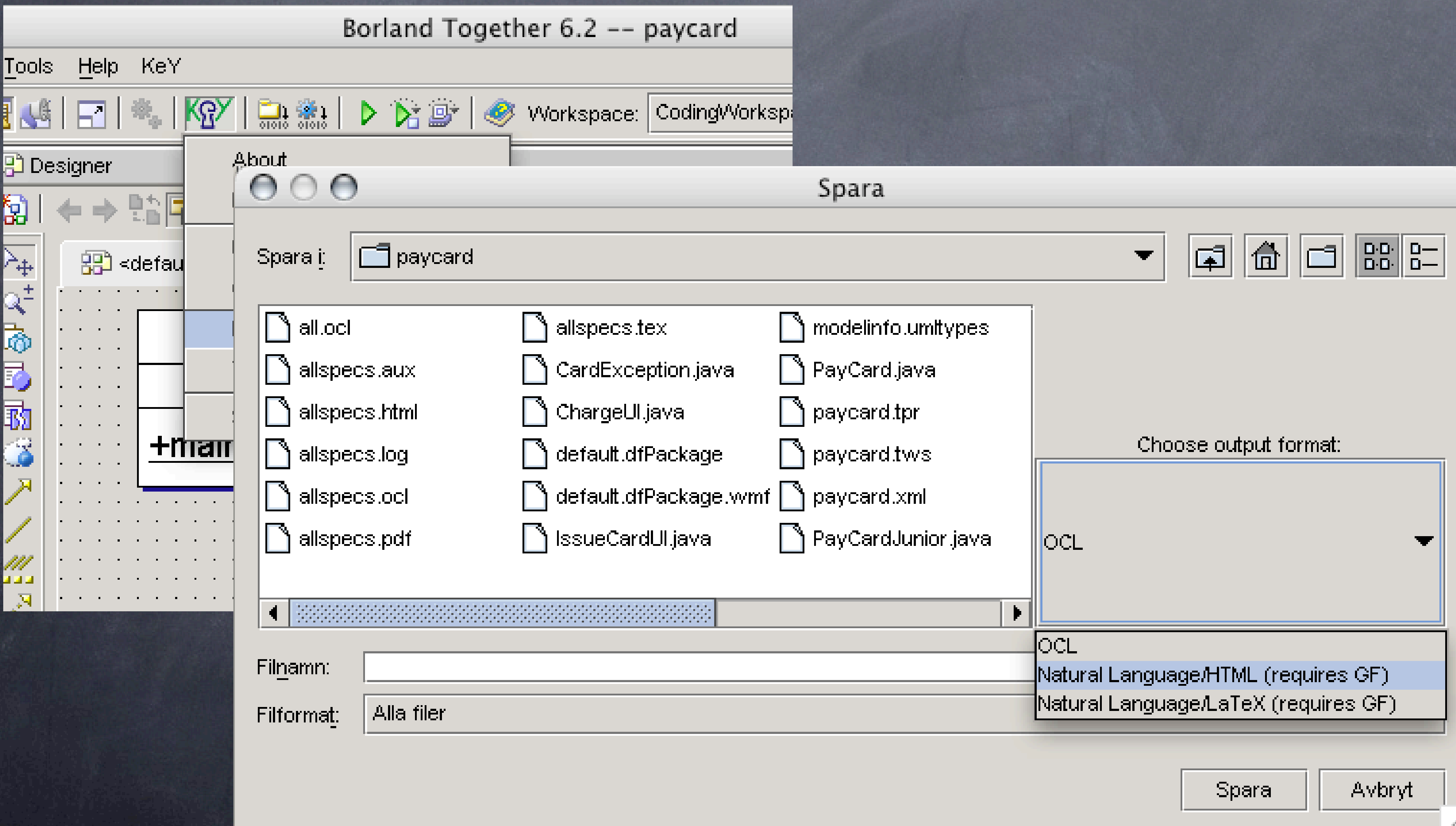
(...)

- if the following conditions are true
  - the try counter is greater than 0
  - *pin* is not equal to null
  - *offset* and *length* are at least 0
  - *offset* plus *length* is at most the size of *pin*
  - the query `arrayCompare (the pin, 0, pin , offset , length )` on `Util` is equal to 0

then this implies that the following conditions are true

  - the result is equal to true
  - this owner PIN is validated
  - the try counter is equal to the maximum number of tries

KeY-integration



GF for OCL and NL

The Grammatical Framework (GF) is a grammar formalism and toolkit [Ranta].

We have a multilingual GF grammar for specifications in OCL and natural language.

GF grammars

- GF grammars separate abstract from concrete syntax.
- Abstract syntax: rules for building syntax trees representing a restricted domain
- Concrete syntax: rules for linearizing syntax trees into expressions of a concrete language
- We can give several concrete syntaxes for one abstract syntax

GF grammars (2)

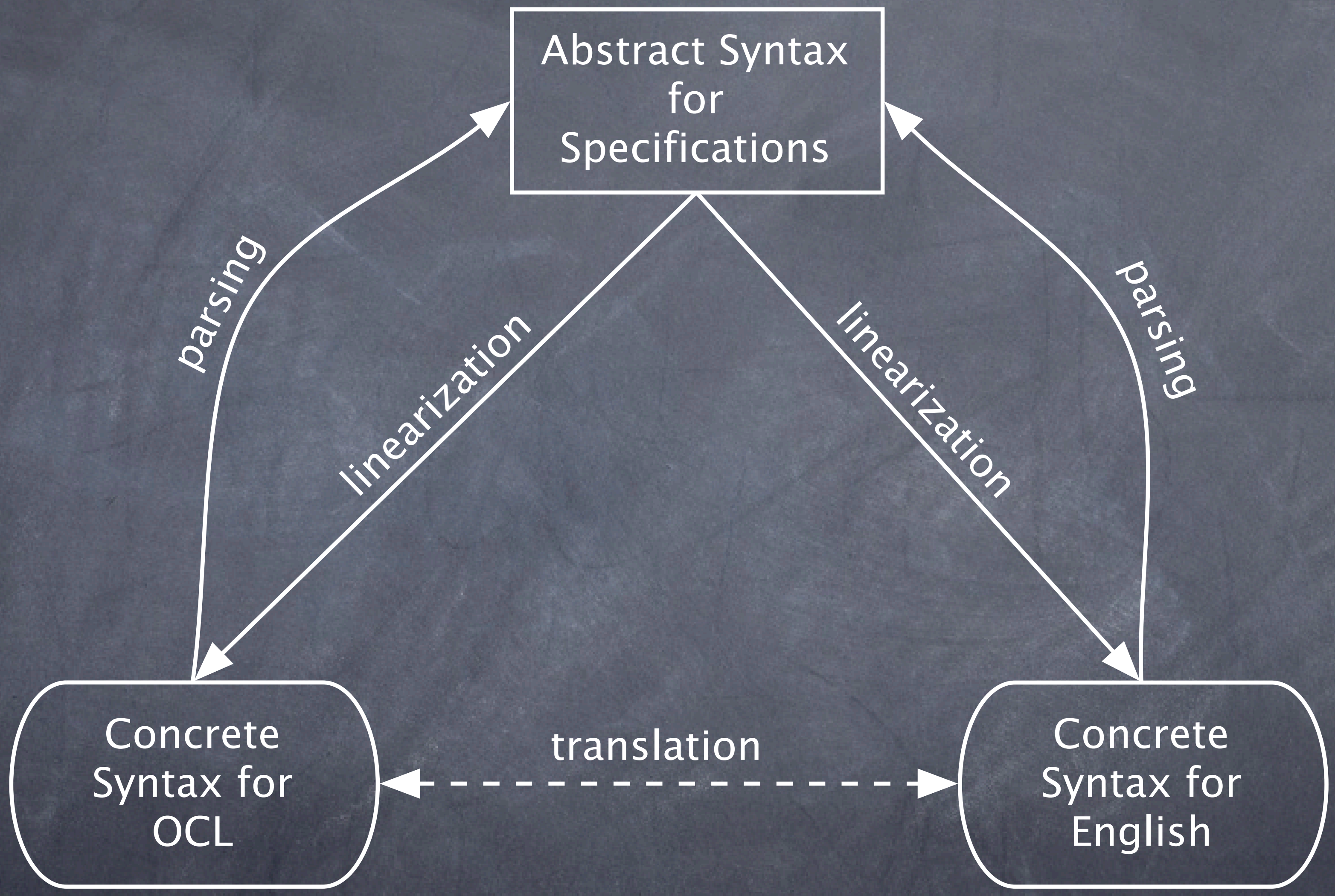
- Abstract syntax is formulated in constructive type theory.
- Concrete syntax gives compositional linearization rules expressed in a restricted functional language.
  - the linearization of a tree is expressed in terms of the linearization of its subtrees — not the subtrees themselves

Functionality provided by GF

- Linearization
- Parsing
- Multilingual, syntax-directed editor

Linking OCL and NL

- Define a GF grammar for software specifications:
  - represent software specifications in the abstract syntax
  - concrete syntaxes for showing specifications in OCL and in NL
- Use GF for:
  - translating OCL to NL
  - multilingual editor for specifications in OCL and NL



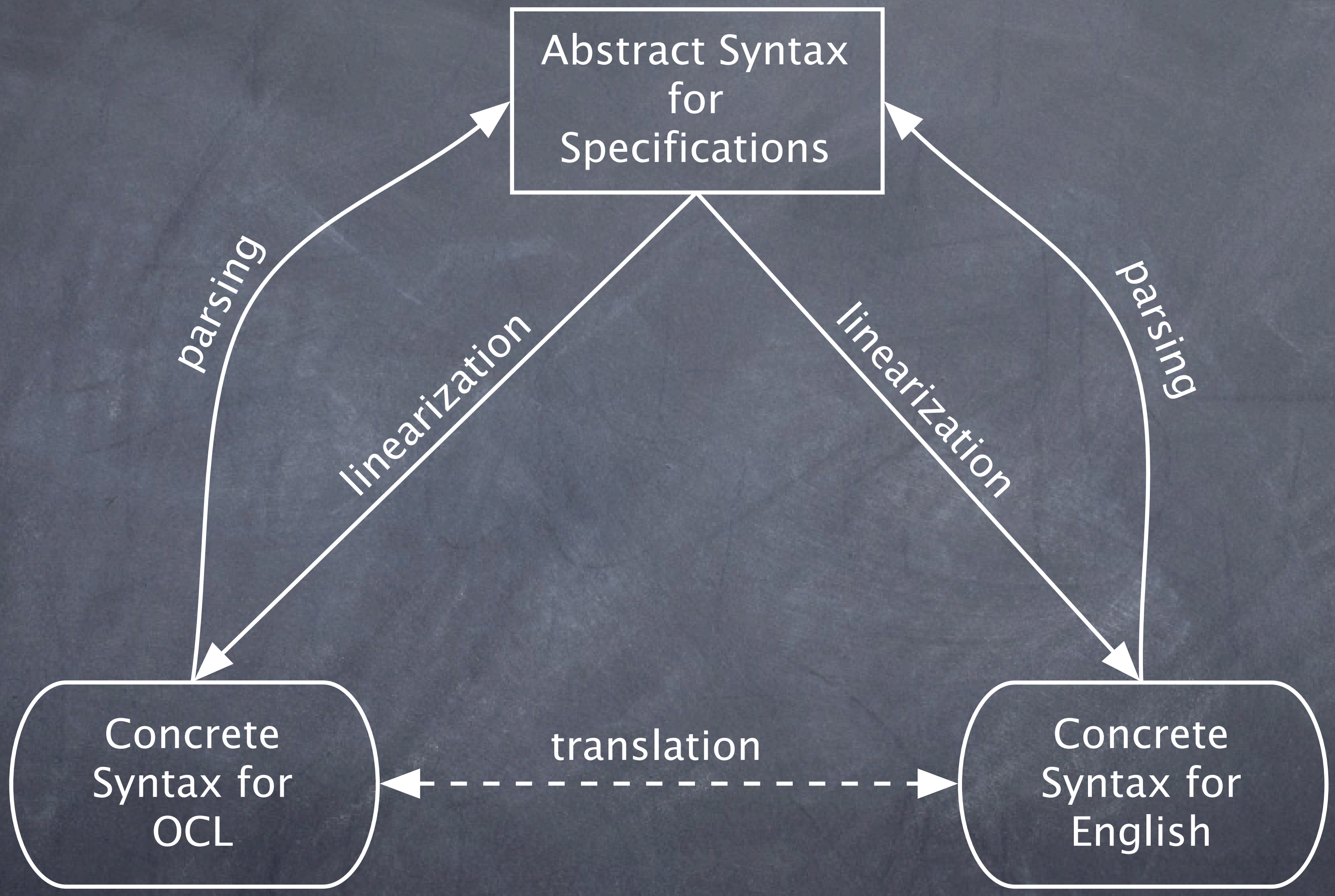
Abstract Syntax for Specifications
parsing
linearization
linearization
parsing
Concrete Syntax for OCL
translation
Concrete Syntax for English

Abstract Syntax

- Semantic representation of specifications
- Compromise between OCL and NL (interlingua)
- Ensures correctness of typing and variable bindings (using dependent types, higher order abstract syntax)

Concrete Syntax

- Present specifications in OCL and NL (English, German)
- We make use of the GF resource grammar library:
  - linguistically motivated types and functions
  - raises the level of abstraction in concrete syntax
  - common interface for 7 languages


Abstract Syntax for Specifications
parsing
linearization
linearization
parsing
Concrete Syntax for OCL
translation
Concrete Syntax for English

Extensions

To handle the case study, we improve our GF-based system with e.g.

- formatting
- customizable domain-specific vocabulary

These improvements are partly implemented inside the GF grammar, partly using external programs.

Formatting

- Fonts and structure (variables in italic, bullet lists)
- GF interface module with formatting functions
  - interface used in the GF linearization rules
  - three implementations: no formatting, HTML and LaTeX

Structure

s_1 and s_2 and ... and s_n →

The following conditions hold:

- s_1
- s_2
- ...
- s_n

This can be seen as a transformation of syntax trees.

Tree Transformations

External program for transforming (optimizing) abstract syntax trees:

```
and s1 (and s2 (... and sn-1 sn))
```

$\downarrow$

```
andList (cons s1 (cons s2 (... cons sn nil)))
```

Domain-Specific Concepts

- Each new concept in a class diagram extends the language of specifications
- Program for generating GF grammar modules from class diagrams
- Simple heuristics based on name and type
  - the class OwnerPIN → the noun "owner PIN"
  - the boolean method isValidated() → the predicate "...is validated"

API for Domain-Specific Concepts

The generated GF modules need some by-hand modifications.

We define an API module with common constructions for domain-specific vocabulary.

- The API is used by the grammar generator and when performing by-hand modifications
- It hides the complexity of the rest of the grammar.

Example Customization

The maxTries attribute of the class OwnerPIN

- Automatically generated linearization:

```
lin maxTries = mkSimpleProperty
      (adjCN “max” (strCN “tries”));
```

- After by-hand modification:

```
lin maxTries = mkSimpleProperty (
      ofCN (adjCN “maximum” (strCN “number”))
      (strCN “tries”));
```

KeY Integration

Future work:

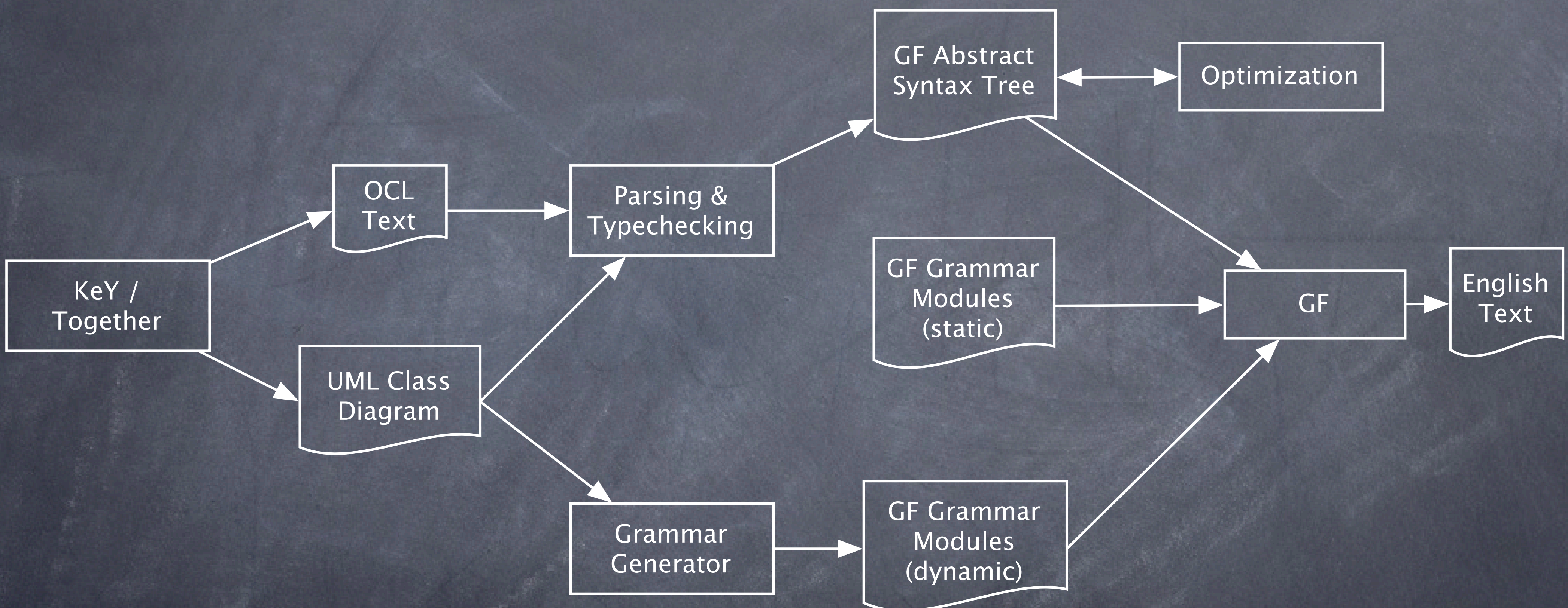
Use javadoc-annotations to customize the translation of classes, attributes, operations and associations.

OCL Parsing & Typechecking

We use an external OCL parser/typechecker:

- work-around for limitation in GF-derived parser for our grammar
- more efficient for large specifications
- special cases of OCL syntax/typing does not need to be described in GF grammar

System Overview



Limitations

- OCL parser & typechecker
- Exporting OCL/UML from KeY/Together
- Domain-specific concepts for German

Conclusion

- We translate non-trivial OCL specifications to NL which is acceptable to a human reader.
- A multilingual GF grammar is complemented with grammar generation and syntax tree transformations.
- The translation can be customized without requiring GF expertise.