Text Mining for Software Engineering

René Witte

Faculty of Informatics
Institute for Program Structures and Data Organization (IPD)
Universität Karlsruhe (TH), Germany

Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada

http://rene-witte.net

14.05.2007

René Witte

Research Interests

Pre-PhD (?–2002): Databases, Information Systems, Fuzzy Theory
PhD on Architecture of Fuzzy Information Systems

Post-PhD (2002–now): Text Mining, NLP, Semantic Web

Text Mining

Deal with unstructured documents written in natural languages:

- newspaper/newswire articles
- biomedical research papers
- encyclopedia on building architecture
- software engineering documents

René Witte

Research Interests

Pre-PhD (?–2002): Databases, Information Systems, Fuzzy Theory
PhD on Architecture of Fuzzy Information Systems

Post-PhD (2002–now): Text Mining, NLP, Semantic Web

Text Mining

Deal with unstructured documents written in natural languages:

- newspaper/newswire articles
- biomedical research papers
- encyclopedia on building architecture
- software engineering documents

Source Code vs. Documentation

A typical problem...

Source Code

```
public class OwlExporter
implements ProcessingResource {
...
}
```

Documentation

The class OwlExporter implements the interface LanguageResource

Recovery of Traceability Links

Background

Traceability links help software engineers understand the relations and dependencies among various software artifacts (e.g., source code, documentation).

Challenge

Links between different artifacts often get lost during the development process, for various reasons:

- Difference in languages (natural language vs. source code)
- Difference in abstraction level (design or requirements vs. implementation)
- Maintanance of links is typically not enforced
- Lack of adequate (semi-automatic) tool support for creating and maintaining links

Recovery of Traceability Links

Background

Traceability links help software engineers understand the relations and dependencies among various software artifacts (e.g., source code, documentation).

Challenge

Links between different artifacts often get lost during the development process, for various reasons:

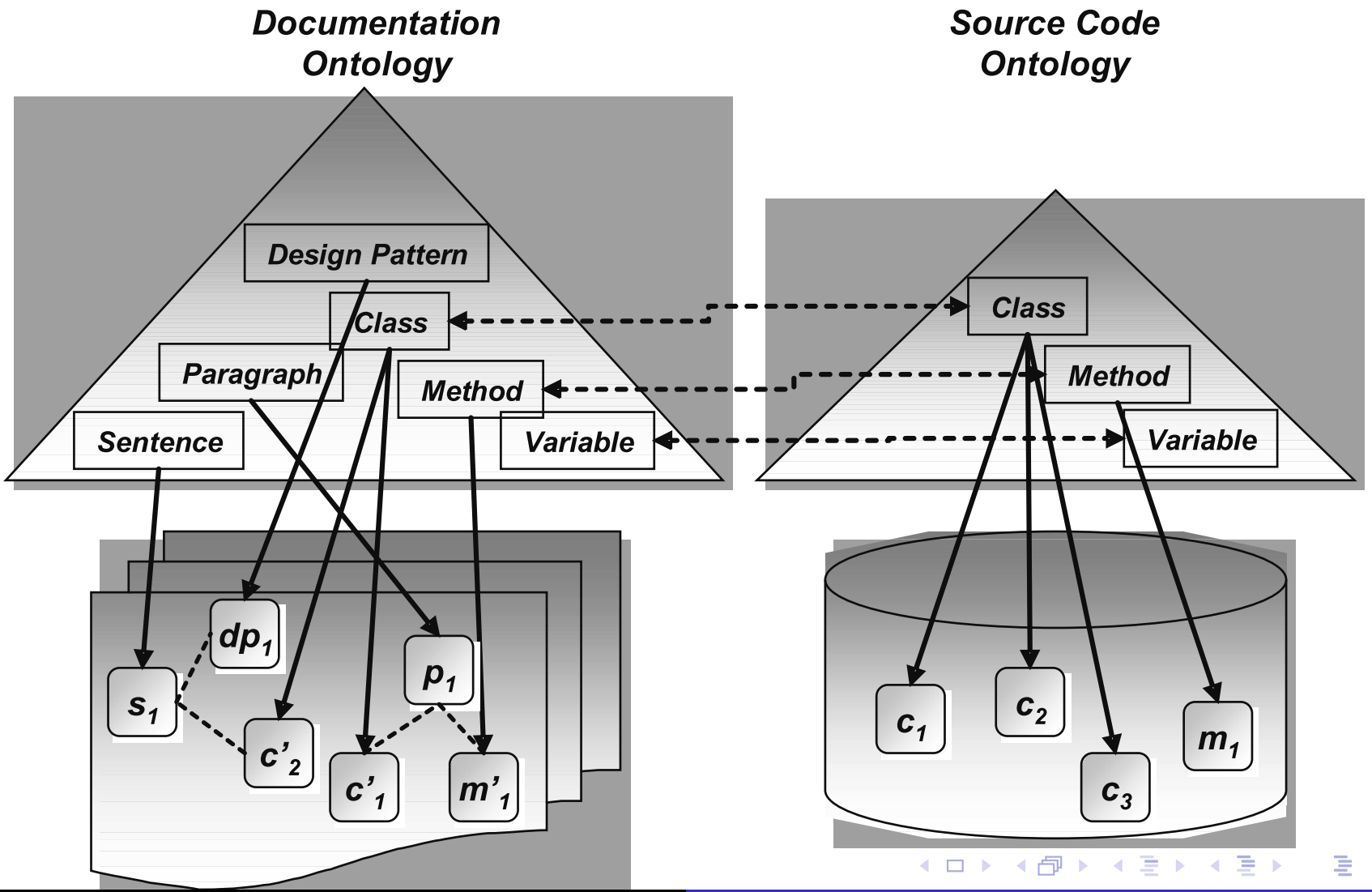
- Difference in languages (natural language vs. source code)
- Difference in abstraction level (design or requirements vs. implementation)
- Maintanance of links is typically not enforced
- Lack of adequate (semi-automatic) tool support for creating and maintaining links

Ontology-Based Approach

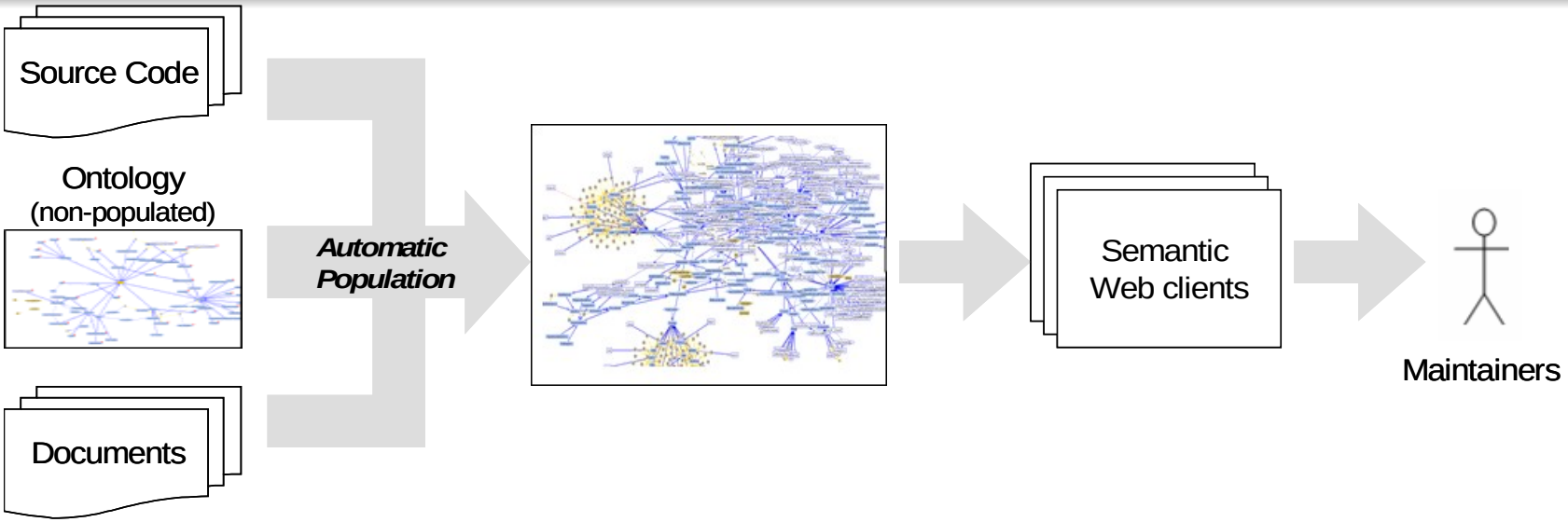
Solution

- Automatic recovery of traceability links
- Use an *ontology* as a single data model for knowledge concerning both source code and documentation artifacts
- Instance information is extracted from source code using compilers and static code analysis
- Likewise, instance information can also be obtained from documents using text mining
- The resulting ontologies can be aligned on the class level and linked or merged to provide traceability (and other new features)

Ontology Aligment: Code and Document Instances



Applications in Software Engineering



Use Cases

- **Architectural Recovery.** Comprehend and maintain large-scale architectures when restructuring code.
- **Security Analysis.** Identify security concerns in source code through ontology queries and reasoning.
- **Recovery of Traceability Links.** Connect code with its corresponding documentation.

Software Ontology

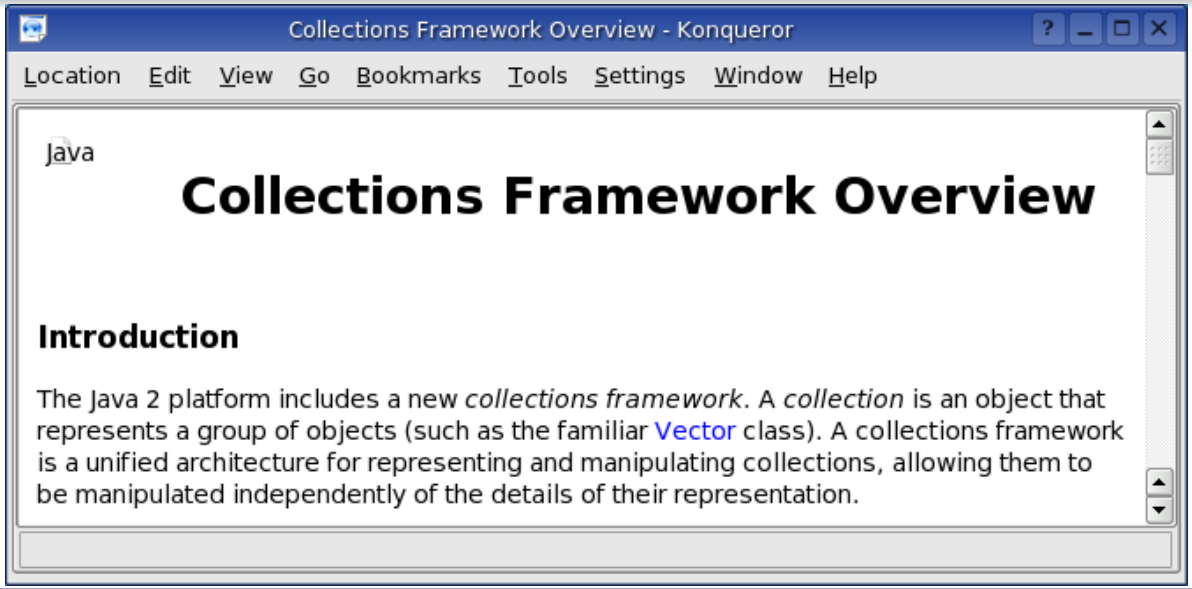
Source Code Sub-Ontology

- Capture major concepts of (object-oriented) programing languages (Class, Variable, Method, etc.)
- Concepts with a direct mapping to source code elements
  $\Rightarrow$ can be automatically discovered by a Java compiler

Documentation Sub-Ontology

- Concepts that can be discovered in software documents:
  - **Programming:** languages, algorithms, data structures
  - **Design:** design patterns and software architectures
  - **Document-specific:** sentences, NPs, coreference chains
- The documentation ontology and source code ontology share many concepts from the programming language domain
- allows us to establish links between source code and documentation

The Software Text Mining System

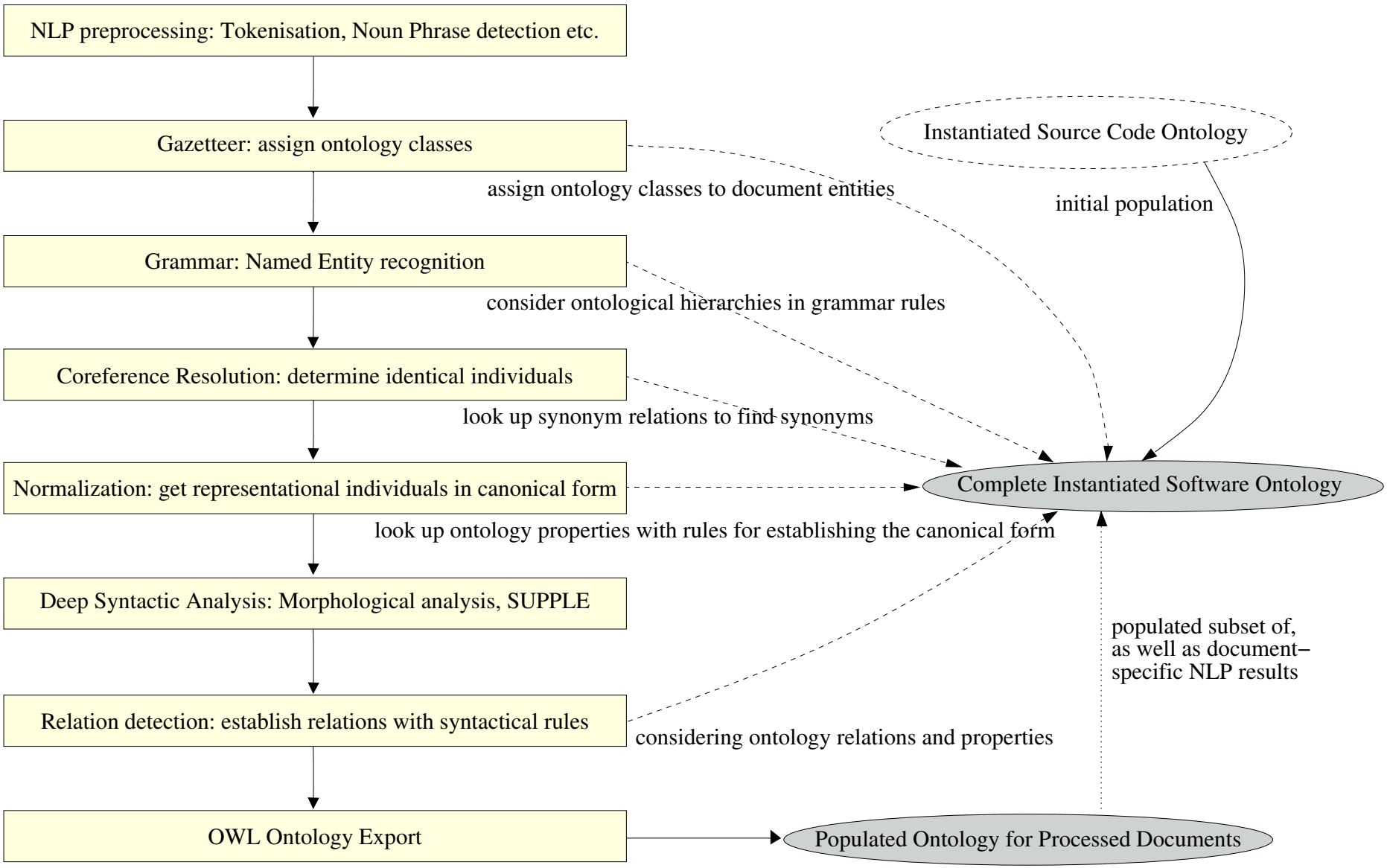


Overview

Input: Software documents written in natural language (currently, English)

Processing: Ontology-based natural language processing to extract semantic knowledge

Output: OWL-DL software ontology, populated with instances detected in documents

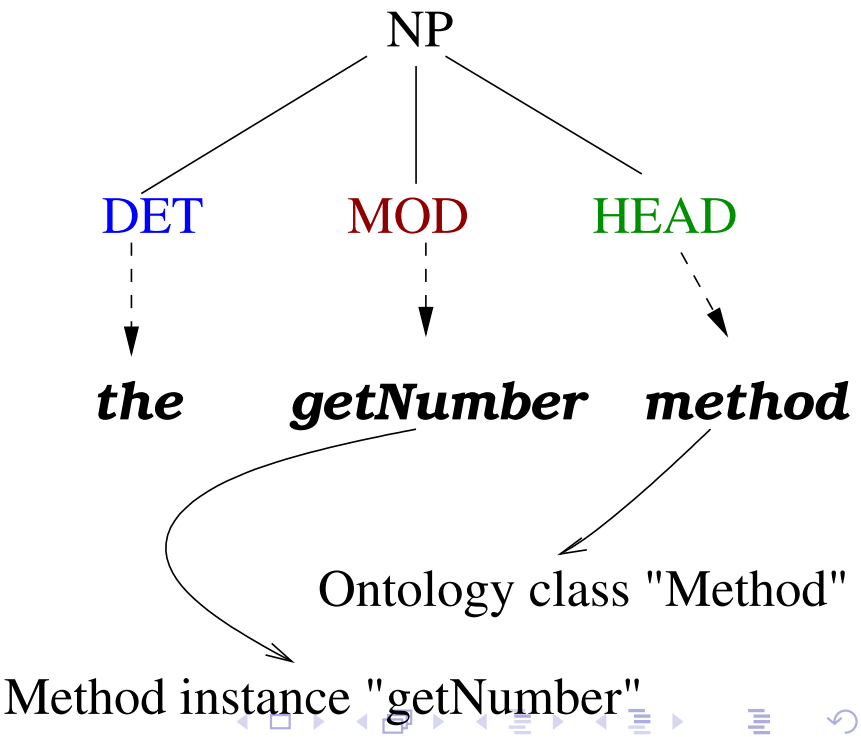

NLP preprocessing: Tokenisation, Noun Phrase detection etc.
Gazetteer: assign ontology classes
assign ontology classes to document entities
Grammar: Named Entity recognition
consider ontological hierarchies in grammar rules
Coreference Resolution: determine identical individuals
look up synonym relations to find synonyms
Normalization: get representational individuals in canonical form
look up ontology properties with rules for establishing the canonical form
Deep Syntactic Analysis: Morphological analysis, SUPPLE
Relation detection: establish relations with syntactical rules
considering ontology relations and properties
OWL Ontology Export
Instantiated Source Code Ontology
initial population
Complete Instantiated Software Ontology
populated subset of,
as well as document–
specific NLP results
Populated Ontology for Processed Documents

Named Entity (NE) Detection

Example

"...that the getNumber method is used ..."

1. OntoGazetteer: Find lexical occurrences of software artifacts
   - "method" is in the "Method" class of the ontologie
2. Perform NP chunking based (mainly) on POS tags
3. Ontology-aware grammar rules (JAPE) to combine both



Relations in Software Documents

Motivation

- Find relations between entites (e.g., <class> *implements* <interface>, <variable> *declared_in* <method>)

Example

"Both the batch and the interactive TestRunner require that the Test class provides a static suite() method."

Approach

- Grammar rules (JAPE transducer)
- Deep syntactic analayis (SUPPLE parser)
- Ontology filter for semantically correct relations

Relations in Software Documents

Motivation

- Find relations between entites (e.g., <class> *implements* <interface>, <variable> *declared_in* <method>)

Example

"Both the batch and the interactive TestRunner require that the Test class provides a static suite() method."

Approach

- Grammar rules (JAPE transducer)
- Deep syntactic analayis (SUPPLE parser)
- Ontology filter for semantically correct relations

Relations in Software Documents

Motivation

- Find relations between entites (e.g., <class> *implements* <interface>, <variable> *declared_in* <method>)

Example

"Both the batch and the interactive TestRunner require that the Test class provides a static suite() method."

Approach

- Grammar rules (JAPE transducer)
- Deep syntactic analayis (SUPPLE parser)
- Ontology filter for semantically correct relations

Automatic Relation Detection

Grammar-based Relation Detection

- Relations defined in ontology and detected through OntoGazetter
- VG chunker module to find verb groups
- hand-crafted grammar rules: *<entity>* *<relation>* *<entity>*

Relationserkennung durch Syntaxanalyse

- SUPPLE bottom-up parser
- extract predicate-argument structures from the resulting parse

Relation Filtering

- Check detected relations for semantic consistency using the ontology
- E.g. *"variable" <implements> "class"* is not valid

Automatic Relation Detection

Grammar-based Relation Detection

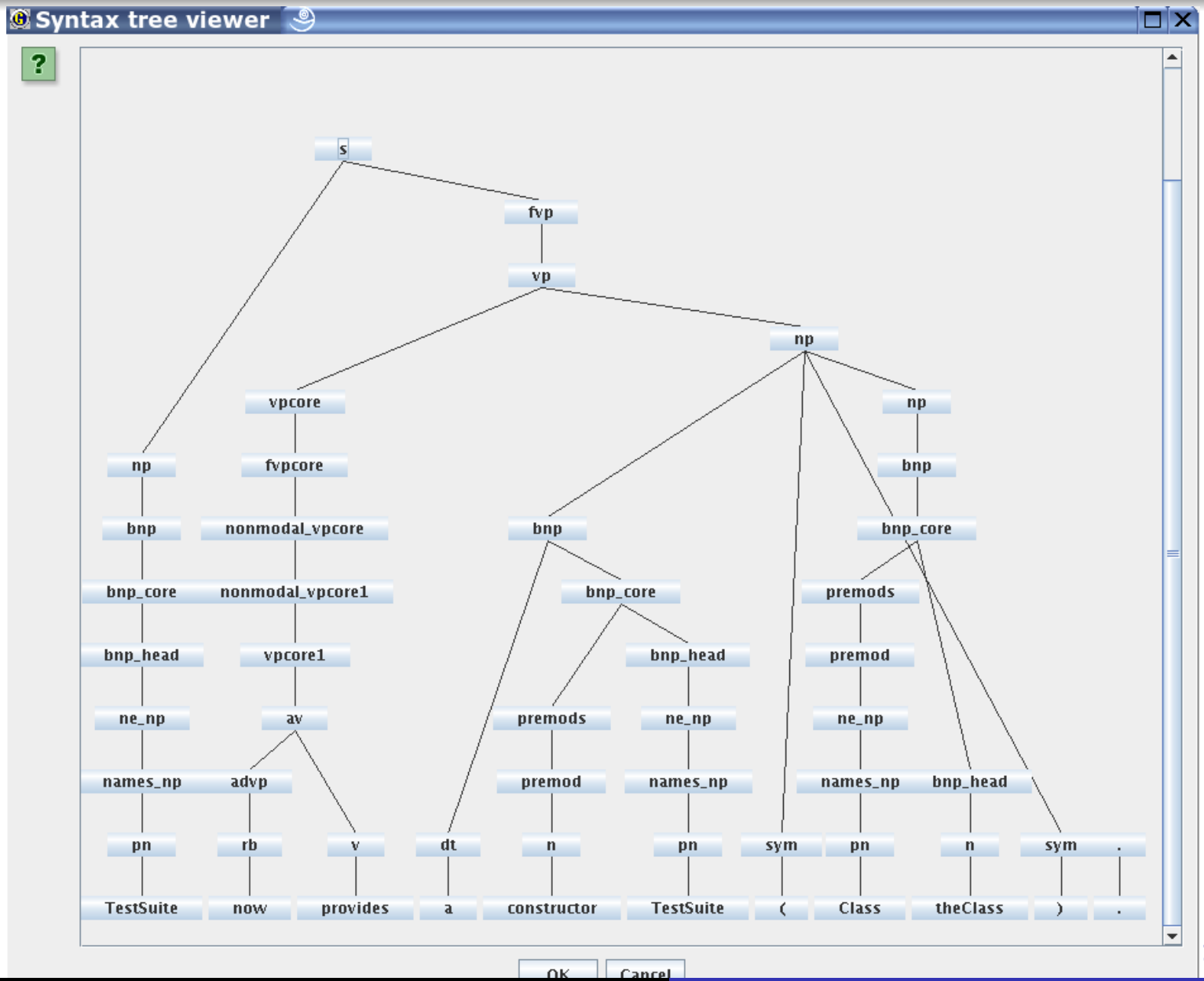
- Relations defined in ontology and detected through OntoGazetter
- VG chunker module to find verb groups
- hand-crafted grammar rules: <*entity*> <*relation*> <*entity*>

Relationserkennung durch Syntaxanalyse

- SUPPLE bottom-up parser
- extract predicate-argument structures from the resulting parse

Relation Filtering

- Check detected relations for semantic consistency using the ontology
- E.g. *"variable"* <*implements*> *"class"* is not valid


Syntax tree viewer
?
s
fvp
vp
np
vpcore
np
np
fvpcore
bnp
bnp
nonmodal_vpcore
bnp
bnp_core
bnp_core
nonmodal_vpcore1
bnp_core
premods
bnp_head
vpcore1
bnp_head
premod
ne_np
av
premods
ne_np
ne_np
names_np
advp
premod
names_np
names_np
bnp_head
pn
rb
v
dt
n
pn
sym
pn
n
sym
.
TestSuite
now
provides
a
constructor
TestSuite
(
Class
theClass
)
.
OK
Cancel

Coreference Resolution and Normalization

Coreference Resolution

Build *coreference chains* using a number of nominal and pronominal heuristics developed for the software domain.

- E.g., *the TestRunner class is implemented, this class is used by...* => Chain: ('the TestRunner class', 'this class')

Entity Normalization

Detected named entites have to be *normalized* for ontology population

- **Text:** *the suite() method*;
- **Normalized:** *suite*

→ achieved through lexical normalization rules, stored in the ontology with their corresponding classes.

Coreference Resolution and Normalization

Coreference Resolution

Build *coreference chains* using a number of nominal and pronominal heuristics developed for the software domain.

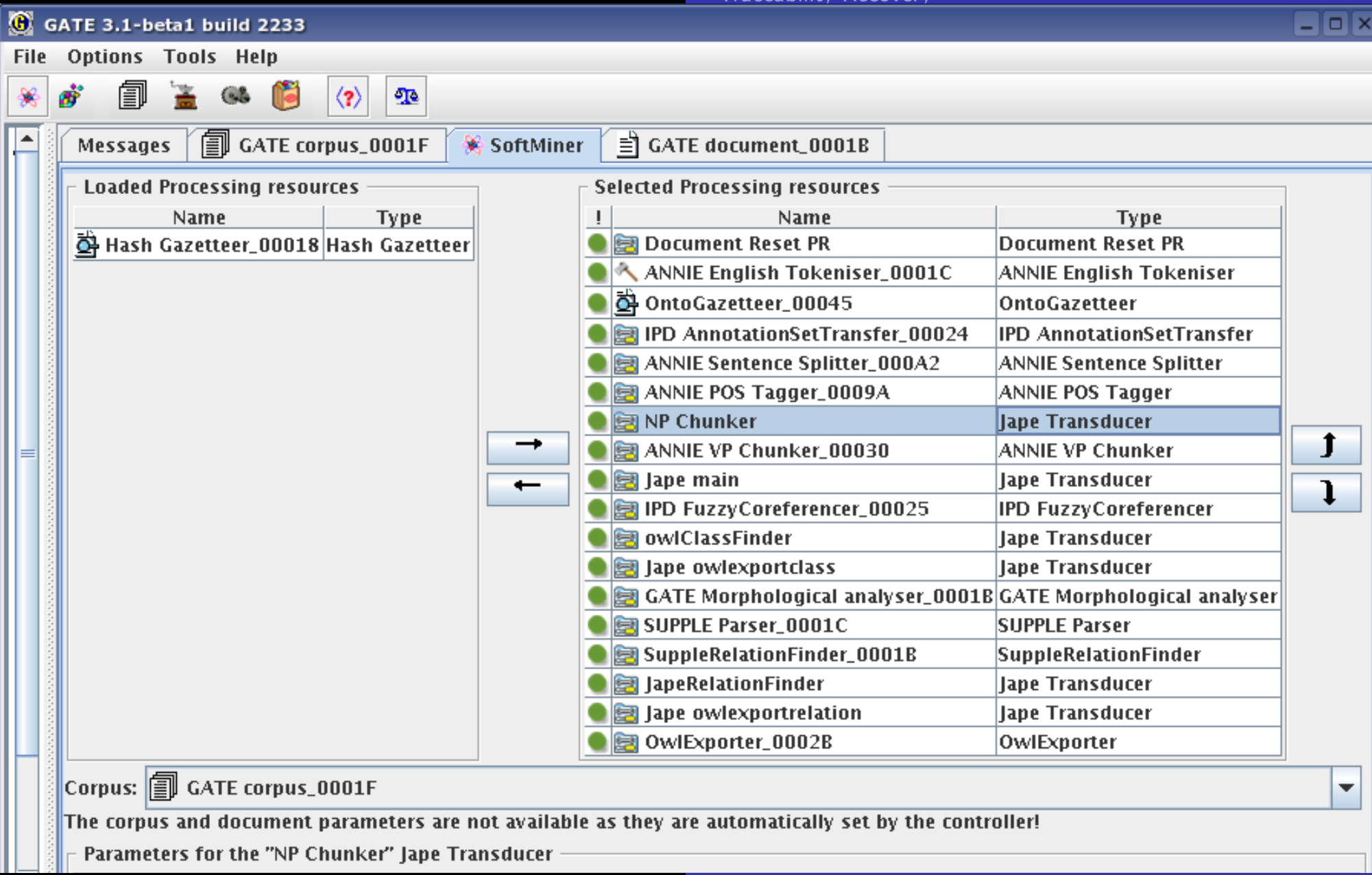
- E.g., *the TestRunner class is implemented, this class is used by...* => Chain: ('the TestRunner class', 'this class')

Entity Normalization

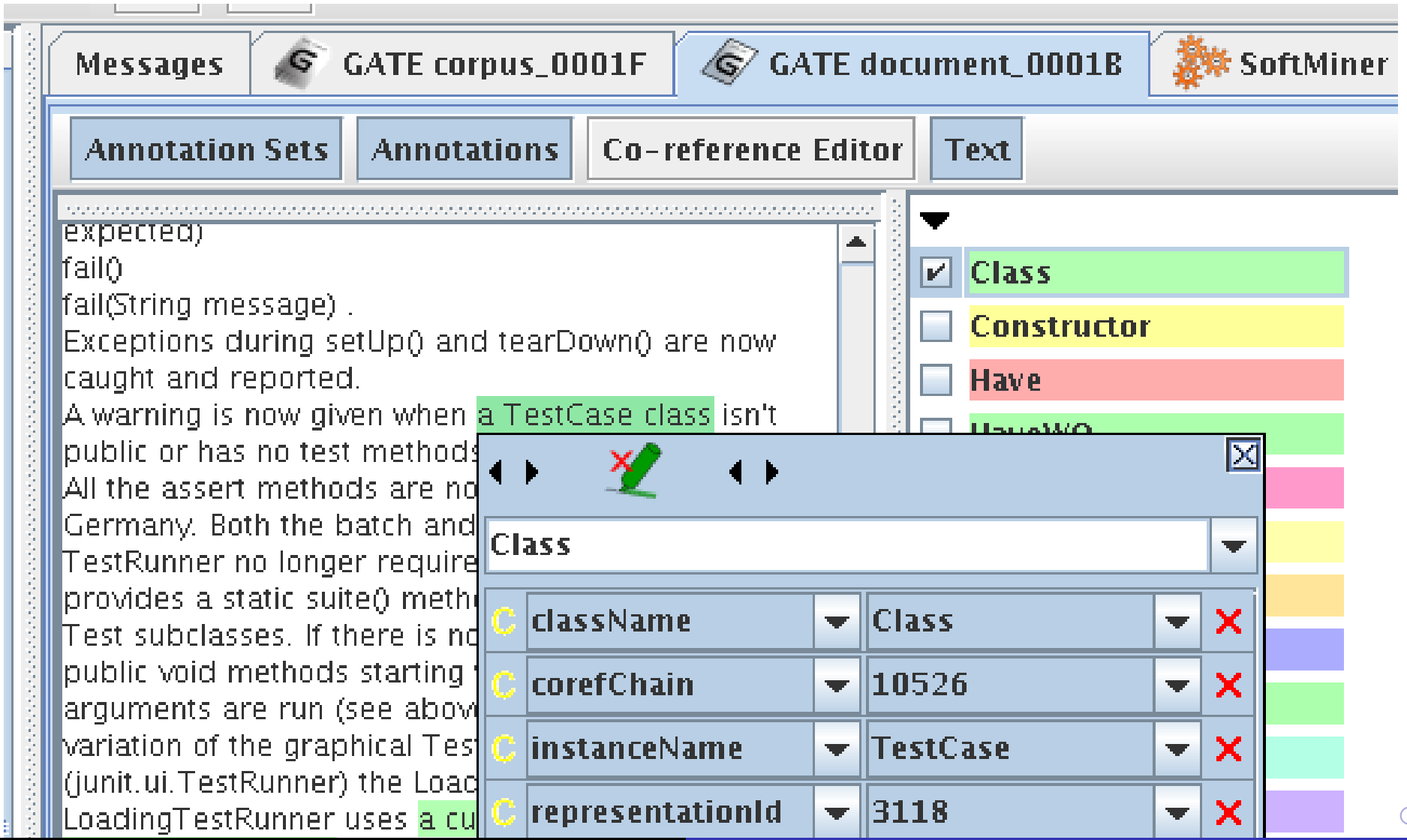
Detected named entites have to be *normalized* for ontology population

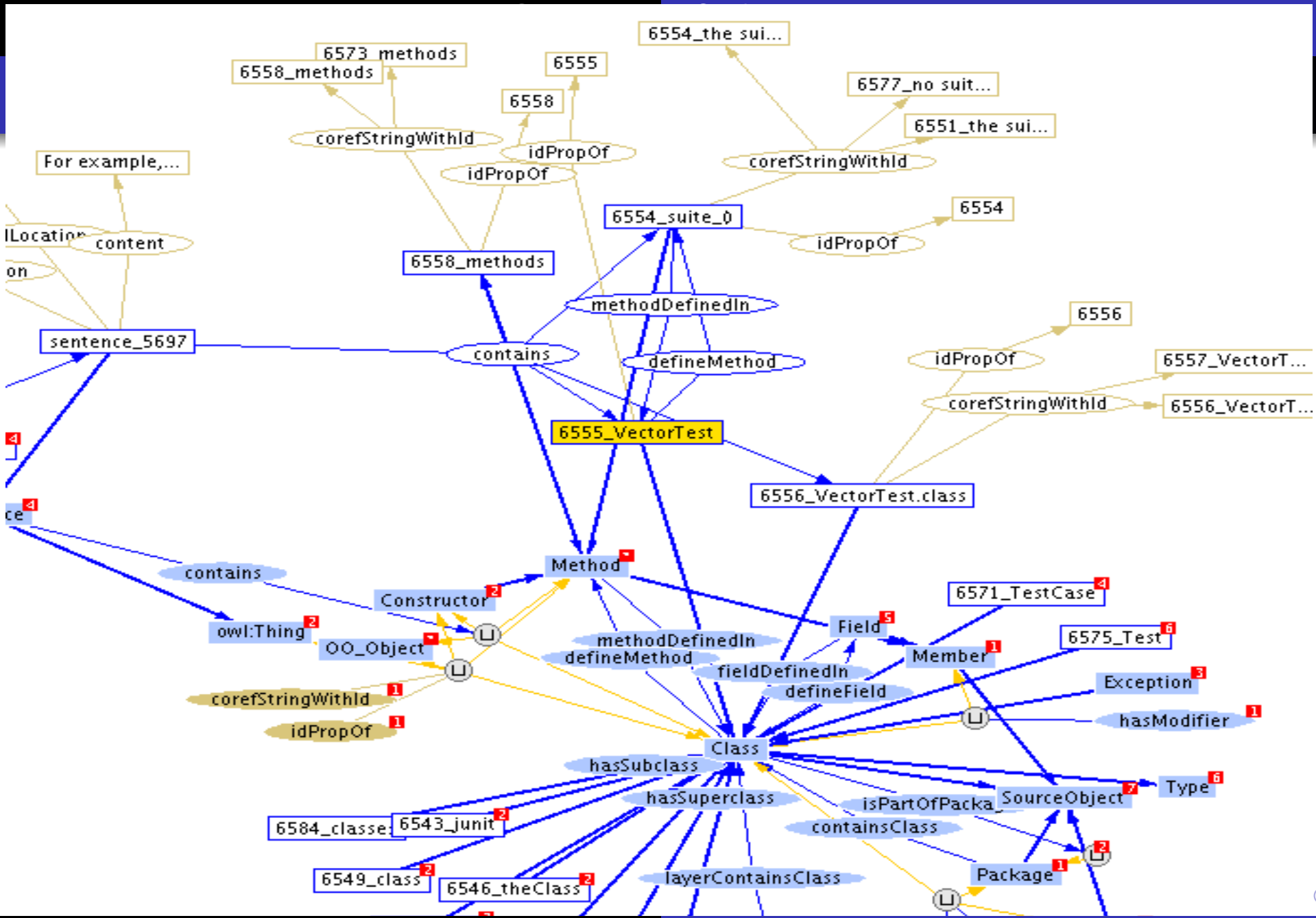
- **Text:** *the suite() method*;
- **Normalized:** *suite*

→ achieved through lexical normalization rules, stored in the ontology with their corresponding classes.

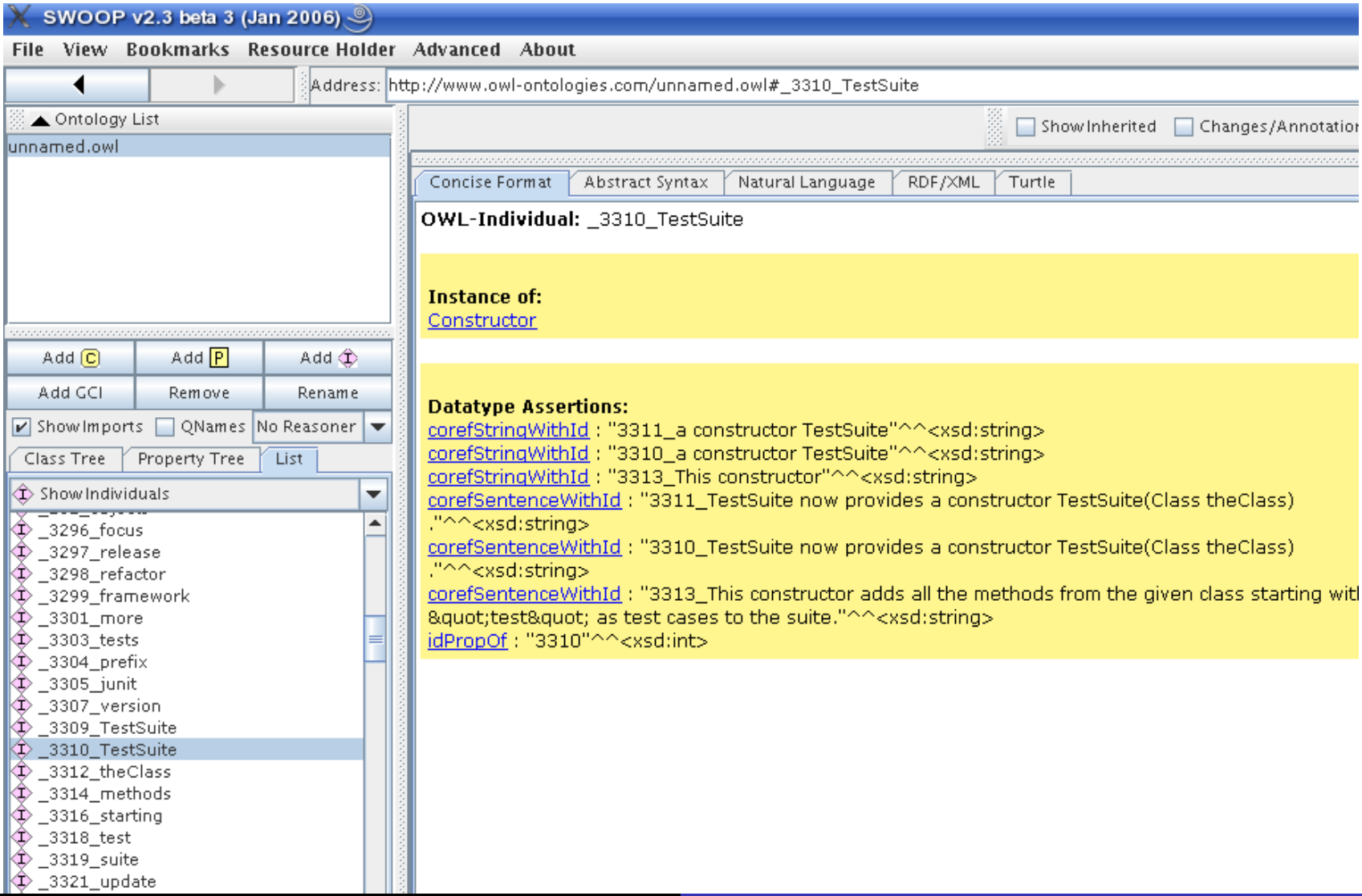

GATE 3.1-beta1 build 2233
File Options Tools Help
Messages
GATE corpus_0001F
SoftMiner
GATE document_0001B
Loaded Processing resources
Name
Type
Hash Gazetteer_00018
Hash Gazetteer
Selected Processing resources
!
Name
Type
Document Reset PR
Document Reset PR
ANNIE English Tokeniser_0001C
ANNIE English Tokeniser
OntoGazetteer_00045
OntoGazetteer
IPD AnnotationSetTransfer_00024
IPD AnnotationSetTransfer
ANNIE Sentence Splitter_000A2
ANNIE Sentence Splitter
ANNIE POS Tagger_0009A
ANNIE POS Tagger
NP Chunker
Jape Transducer
ANNIE VP Chunker_00030
ANNIE VP Chunker
Jape main
Jape Transducer
IPD FuzzyCoreferencer_00025
IPD FuzzyCoreferencer
owlClassFinder
Jape Transducer
Jape owlexportclass
Jape Transducer
GATE Morphological analyser_0001B
GATE Morphological analyser
SUPPLE Parser_0001C
SUPPLE Parser
SuppleRelationFinder_0001B
SuppleRelationFinder
JapeRelationFinder
Jape Transducer
Jape owlexportrelation
Jape Transducer
OwlExporter_0002B
OwlExporter
Corpus:
GATE corpus_0001F
The corpus and document parameters are not available as they are automatically set by the controller!
Parameters for the "NP Chunker" Jape Transducer

NE Detection & Normalization Example




6554_the sui...
6573_methods
6558_methods
6555
6558
6577_no suit...
6551_the sui...
corefStringWithId
idPropOf
idPropOf
corefStringWithId
For example,...
6554_suite_0
6554
content
idPropOf
6558_methods
methodDefinedIn
6556
sentence_5697
contains
defineMethod
idPropOf
6557_VectorT...
corefStringWithId
6556_VectorT...
6555_VectorTest
6556_VectorTest.class
contains
Method
Constructor
6571_TestCase
owl:Thing
OO_Object
Field
6575_Test
methodDefinedIn
defineMethod
Member
fieldDefinedIn
defineField
Exception
corefStringWithId
idPropOf
hasModifier
Class
hasSubclass
Type
hasSuperclass
SourceObject
isPartOfPacka
6584_classe
6543_junit
containsClass
6549_class
6546_theClass
layerContainsClass
Package

Navigating a populated ontology with SWOOP



Automatic Traceability Recovery

Results so far

We now have two instantiated OWL ontologies:

- Source code ontology (from software analysis)
- Documentation ontology (through text mining)

Next Step

We now have to link the two ontologies to find information concerning an entity from both sides

- For example, a "class" appears in both ontologies

Solution: Ontology Alignment

Classes appearing in both ontologies are candidates for alignment;

- Instances from those classes that share the same name (or certain properties) are assumed to be equal

Automatic Traceability Recovery

Results so far

We now have two instantiated OWL ontologies:

- Source code ontology (from software analysis)
- Documentation ontology (through text mining)

Next Step

We now have to link the two ontologies to find information concerning an entity from both sides

- For example, a "class" appears in both ontologies

Solution: Ontology Alignment

Classes appearing in both ontologies are candidates for alignment;

- Instances from those classes that share the same name (or certain properties) are assumed to be equal

Automatic Traceability Recovery

Results so far

We now have two instantiated OWL ontologies:

- Source code ontology (from software analysis)
- Documentation ontology (through text mining)

Next Step

We now have to link the two ontologies to find information concerning an entity from both sides

- For example, a "class" appears in both ontologies

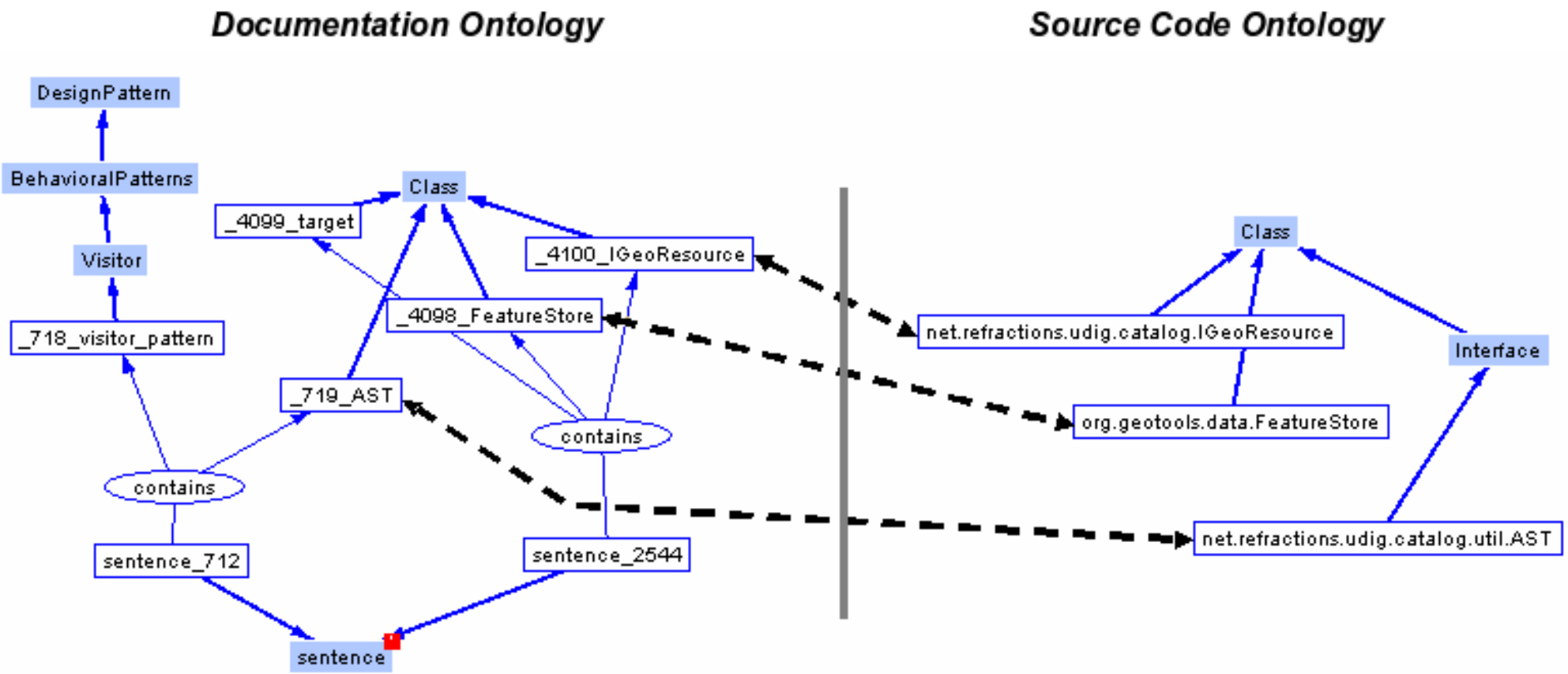
Solution: Ontology Alignment

Classes appearing in both ontologies are candidates for alignment;

- Instances from those classes that share the same name (or certain properties) are assumed to be equal

Traceability Recovery

Analysis of the uDig GIS: Source code and corresponding documentation



Conclusions

NLP and Software Engineering

Dealing with semantics is an emerging topic in software engineering:

- Natural language documents are (almost) completely unused for automated software engineering tasks
- While we cannot really "understand" natural language yet, language technology has matured to a point that makes targeted automated analyses feasible on a large scale

Automatic processing requires shared representation format:

- Ontologies (in OWL-DL) are expressive, standardised (W3C), provide for automated reasoning, and are well supported by tools

Thank You!

Questions?

More information: `http://rene-witte.net`