



From Language-Independent Requirements to Code Based on a Semantic Analysis

Authors:

Mariam MEFTEH

Nadia Bouassida

Hanène Ben Abdallah

Outline

1. Context and Motivations
2. Aim of this work
3. State of the art
4. Our approach for source code extraction from requirements
5. Evaluation
6. Conclusion
7. Future work

Context and motivations



Programming languages in progress



Programmers are forced to update always their knowledge to:

- Recover this progress
- Be able to use them



Context and motivations



- Ideas are almost the same if we express them in several languages.
- They would be valid as long as the language, in which they are expressed, exists and is understood by people.

Context and motivations



- Pegasus : new, naturalistic programming language.
 - ➔ Naturalistic Programming = writing computer programs with the help of a natural language.
- Pegasus accepts instructions written in a semi-natural language.
- Pegasus produces the respective program accordingly.

State of the art

1. Works for Information Extraction from Texts

e.g. the Frame semantics approach [*Fillmore, 1976, 2010*][*Ruppenhofer et al., 2010*]

- They are relevant for specific domains.
- The information of the type of a sentence (e.g., a definition, a statement, an assignment, etc.) cannot be determined by the frame semantics approach

State of the art

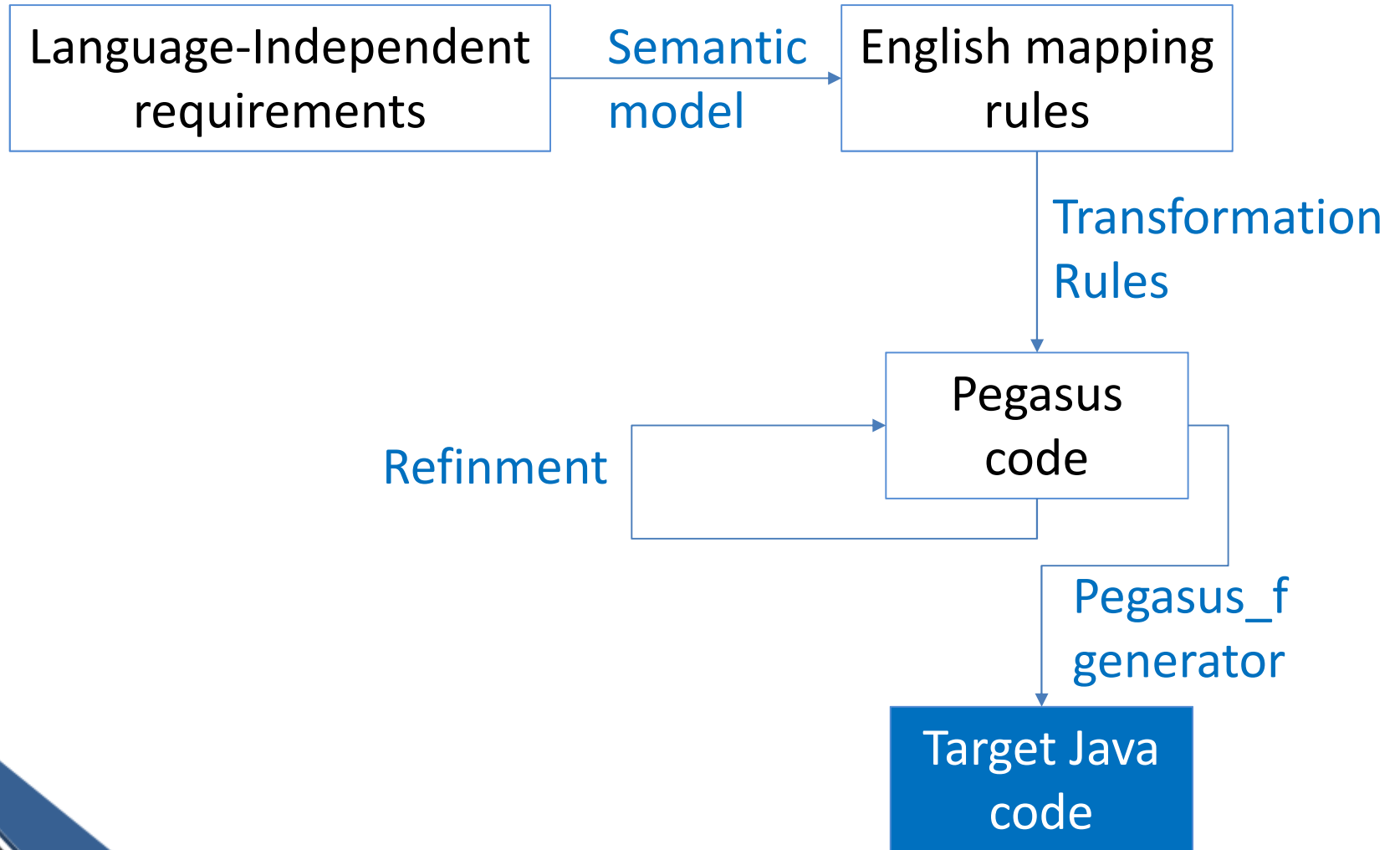
2. Works for Source Code Generation from Textual Requirements Texts

- *[Francu et al., 2011]*: Produce an implementation in the form of methods.
 - Necessity of a manual processing to build a domain model, required by the generator.
- *[Smialek et al., 2012, 2015] [Nowakowski et al., 2013] [Kalnins et al., 2014]*: Transform the requirements, in particular behavior scenarios written in a semi-natural language, into Java code.
 - The use case scenarios must be pre-processed and written in a semi-natural language (RSL)
- *[Liu et al., 2005] [Cozzie et al., 2012] [Ozcan et al., 2004]*: Generate code from texts
 - Use of simple mapping models that map nouns to objects and arguments, verbs to methods and adjectives to attributes
- The majority of the existing approaches treat only requirements written in one single language (English in most cases)

Our approach for source code extraction from requirements

- ✓ In this work, we present our approach and its tool, entitled CRec-tool, for extracting source codes from language-independent requirements
- ✓ Challenges:
 - Accept requirements written, theoretically, in any natural language.
 - Not require a manual transformation of the requirements into the syntax of a specific language.
 - ➔ These two merits rely on the concept of semantic model
 - Rely on this model as a solution to hold (almost) all the semantics of the input requirements written in a purely natural language.

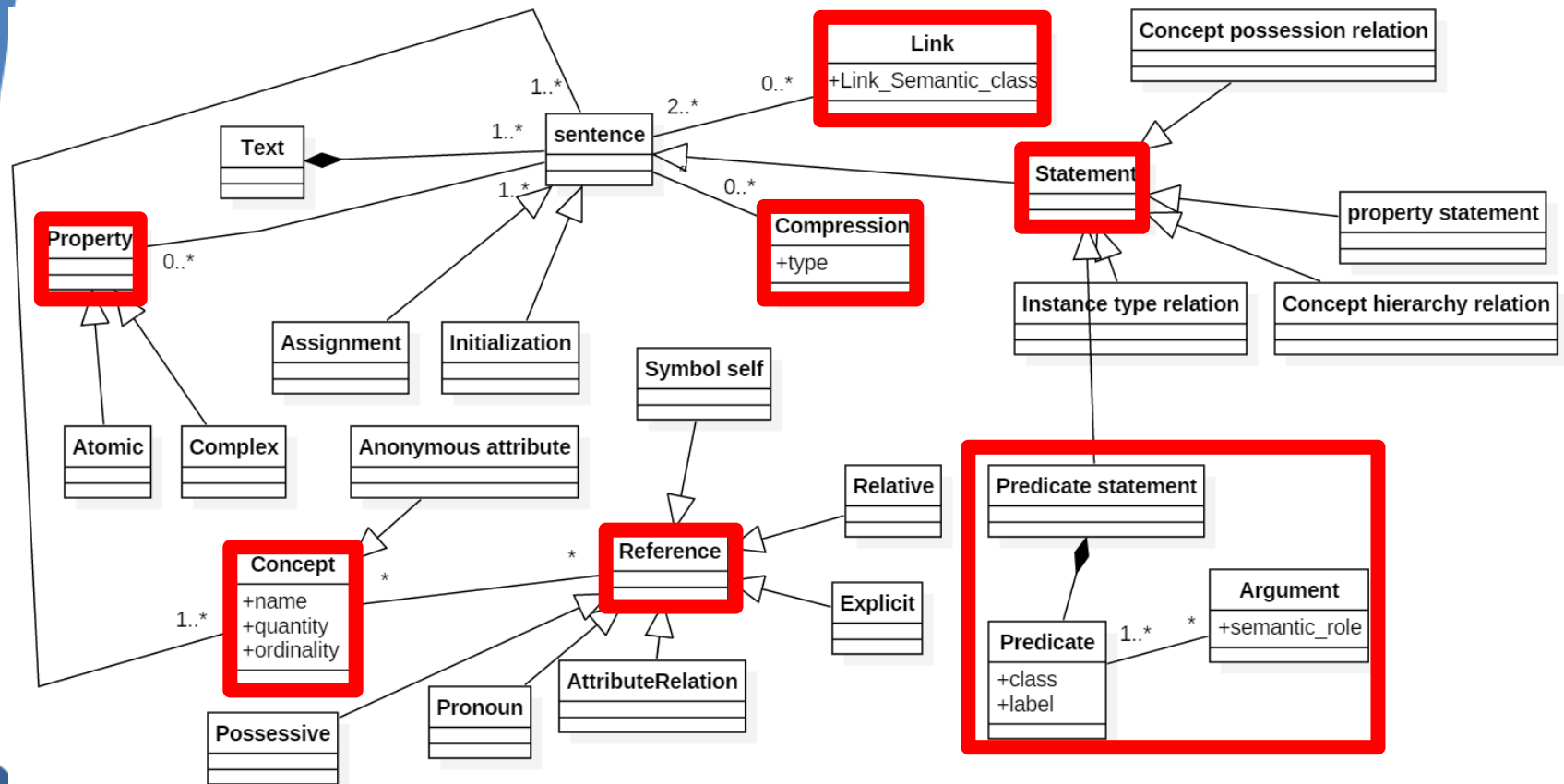
Our approach for source code extraction from requirements



Our approach for source code extraction from requirements

ExtraPL approach: Semantic model-based requirements modelling

Semantic model meta-model



Our approach for source code extraction from requirements

Example: Concept possession relation mapping rule =

```
"(definition/ statement,  
  concept possession relation,  
    (possessor, " possessor_concept " ),  
    (possessed, " possessed concept " ))";
```



For instance, the clauses:

- “A book has a three-letter key” (in English)
 - “Un livre a une cle de trois lettres” (in French)
- ➔ They convey to one common representation within the semant model, as follows:

```
(statement, concept possession relation,  
  (possessor, (book, (quantity/ordinality, abstract))),  
  (possessed, (key, (quantity/ordinality, abstract),  
    three-letter)))
```

Our approach for source code extraction from requirements

The semantic model is able to:

-  Extract relevant information even from quite complex and ambiguous sentences.
-  Treat language-independent texts from which it extract relevant information useful for different purposes (e.g. SPLs, code generation, text translation...)

Our approach for source code extraction from requirements

- CodeRec-tool (Code Recovery tool) implements our approach using the NOOJ environment.
- It allows generating a Pegasus code, as an input to the Pegasus f generator, by starting from requirements written in different and in purely natural languages.
- Actually, this tool accepts the French and the English languages.
 - ✓ Developing a NOOJ syntactic/semantic grammar for each language.
 - ✓ These grammars allow to synthesize mapping rules in English, in cooperation with the predefined NOOJ dictionaries for the English and the French languages, as well as the Google Translate API

Evaluation

1. Goal:

- Showing the ability of our approach in deriving useful Pegasus codes (implicitly good Java codes) from input requirements, written in purely and in several natural languages.
- Examining the conformity degree between our Pegasus codes and those built by Pegasus experts.

2. Subject and Preparation:

- Relying on the use case scenarios belonging to five different domains:
 - Health complaint application
 - The use case “Withdraw cash” belonging to a banking system
 - Go-phone system
 - Crisis management system
 - Game of war cards
 - Emptio application
- Giving them to two Pegasus experts (two natural language processing PhD students from our laboratory, who are familiar to Pegasus programs)

3. **Task:** We asked the Pegasus experts to give us the correct Pegasus codes corresponding to the adopted subjects.
4. **Conduction:** Comparing the experts' Pegasus codes to the corresponding ones generated by our tool by using:
 - The precision and recall metrics in terms of Pegasus concepts, properties and actions.
 - Two other metrics taken from the standard ISO 25020: Completeness and Correctness

Evaluation

Average	Precision	Recall	Comple.	Correct.
Concepts	73,78%	91,60%	90,43%	80,13%
Properties	81,59%	82,66%	78,41%	-
Actions	70,22%	82,41%	76,78%	83,02%

5. Evaluation:

- High average values of the adopted metrics.
- High average values of completeness and correctness.
- ➔ Our tool generates a good number of true positives, vs. a low number of false positives.
- ➔ The code generated by our tool is of a high quality and helps the developers in the programming task by saving their time, and thus money for the companies.

Evaluation

Threats to validity

- The version Pegasus_f of code generator has not yet been finished totally
 - There are still some small improvements to integrate in this project,
 - However, the current version of Pegasus f is powerful and it generates good results, shown in our evaluation.
- A grammar should be created for each integrated language in order to synthesize the mapping rules.
 - However, the creation of this grammar is done only one time.
- Generation of an important number of concepts and actions (and thus Java classes and methods).
 - However, the unnecessary classes and methods generated by our approach will be latter removed by the programmer when revising the generated version of source codes.
- Our approach is dependent from the input requirements.

Conclusion

- New, original approach for extracting source code from requirements written, theoretically, in any and purely natural language.
- The developers will save time because they are not obliged to create the initial classes of the system or to import the required packages.
- Our approach is implemented by the CodeRec-tool.
- Our approach is very simple
 - It does not necessitate any pre-study on a particular language to be used.
 - It accepts language-independent descriptions, understandable even by a non-IT person.
- Our approach is able to be used with many code generators, not necessarily Pegasus_f.

The research that we presented in this paper constitutes a contribution in programming using any and purely natural language thanks to the semantic model.

Future work

- Conduct an evaluation on a larger set of products to confirm the presented results.
- Application of our approach on other code generators, such as the ReDSeeDS tool, which extracts a Java code, following a Model/View/Controller architecture, from use case scenarios written in the RSL language.



Thank you

E-mail: mariem.mefteh.ch@gmail.com