How to Make Legacy Code MDSD–Ready

Robert Walter, Oliver Haase
HTWG Konstanz

16. Oktober 2008

Overview

Overview

Why Does Legacy Code Matter?

- MDSD is about to make the transition from academic sphere to industrial–grade SW development
- migration pathes and integration of legacy code play a key role for its acceptance (or rejection)

Question

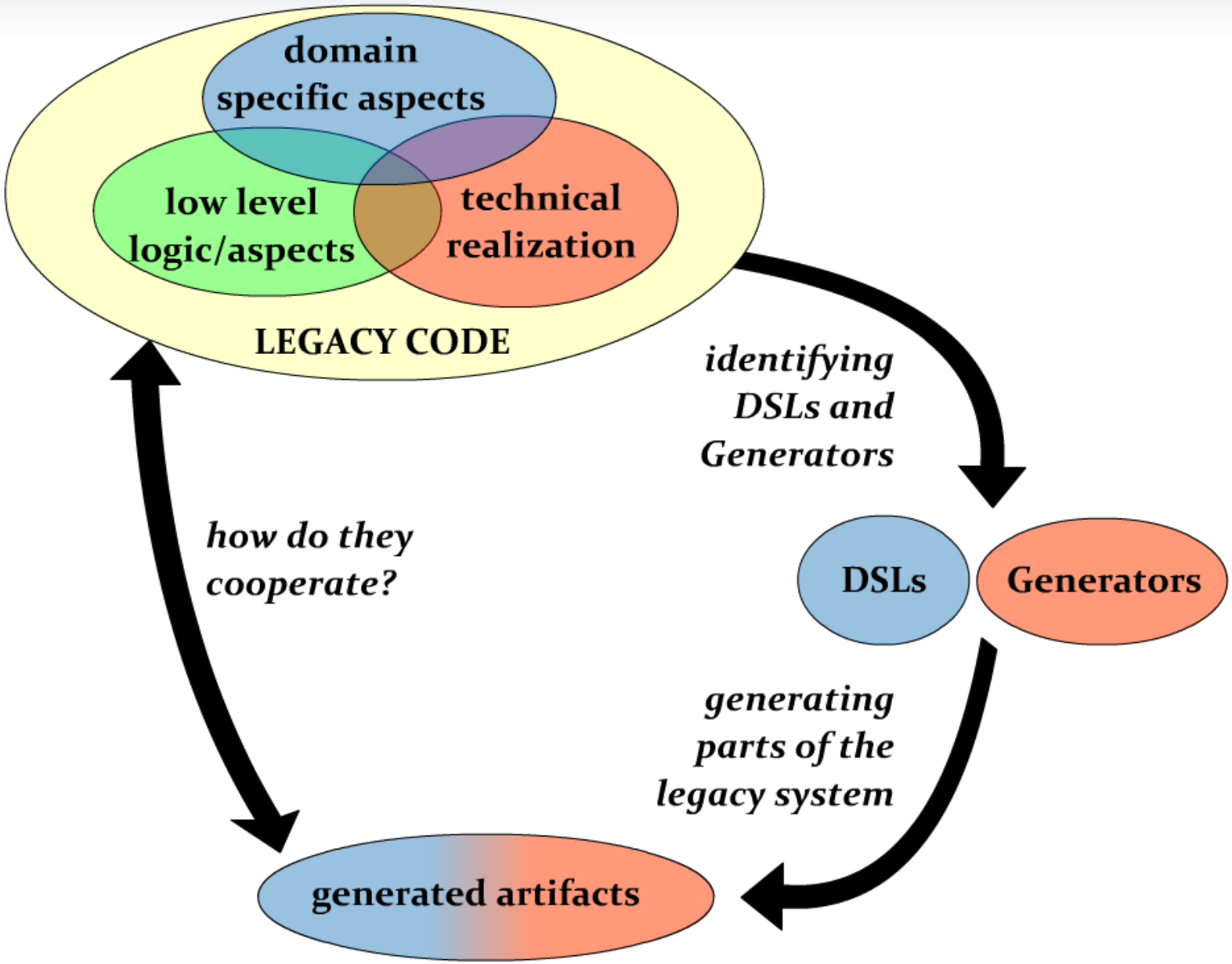
How can legacy systems be integrated in an MDSD approach?

The Challenge

- legacy systems include
  - domain specific aspects ($\rightarrow$ metamodels),
  - technical aspects ($\rightarrow$ tranformation rules) and
  - "low level logic" ($\rightarrow$ manual code),
  - often strongly linked.
- challenge
  - extracting domain specific aspects to build a domain description ($\rightarrow$ DSL(s))
  - developing transformation rules ($\rightarrow$ generator(s))
  - **let the generated artifacts co–work with the not generatable parts of the legacy system**

Conclusion

Legacy systems *somehow* need to be restructured to enable an easier interoperability between generated and hand crafted artifacts.


domain
specific aspects
low level
logic/aspects
technical
realization
LEGACY CODE
identifying
DSLs and
Generators
DSLs
Generators
generating
parts of the
legacy system
generated artifacts
how do they
cooperate?

The Goal

What if the structure of a legacy system is made for cooperative generated and hand crafted code?

- automated restructuring of legacy code
- new structure offers a well defined separation of generated and hand crafted code
- restructuring does not change the systems functionality
- as less as possible manual adjustments need to be done after the restructuring
- having a runnable system at any time of the integration

Overview

The Idea

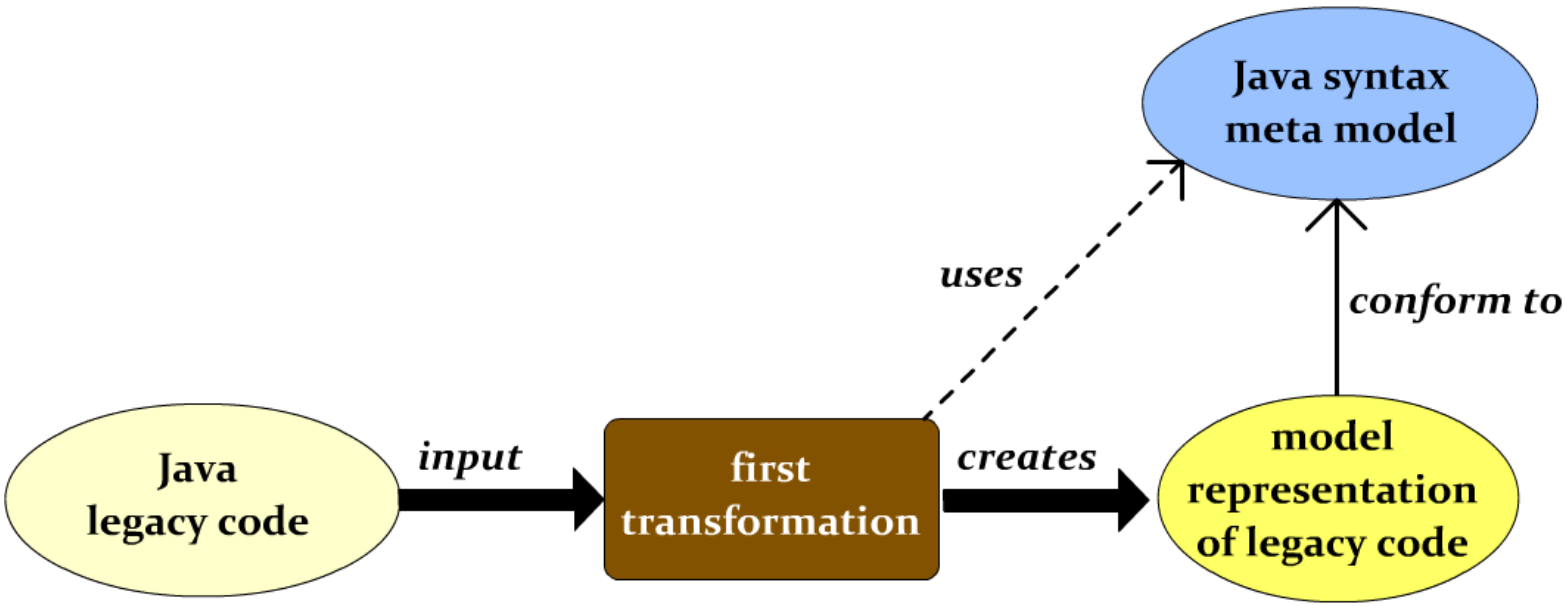
Using MDSD–mechanisms to restructure legacy code:

- two–tiered transformation process
  - first, transform the code base to a specific representation (a model) which conforms to a given meta model
  - second, the model is "transformed back" to code which features the necessary structure
- using an object oriented pattern to achive the interoperability of generated and h/c code

Attention

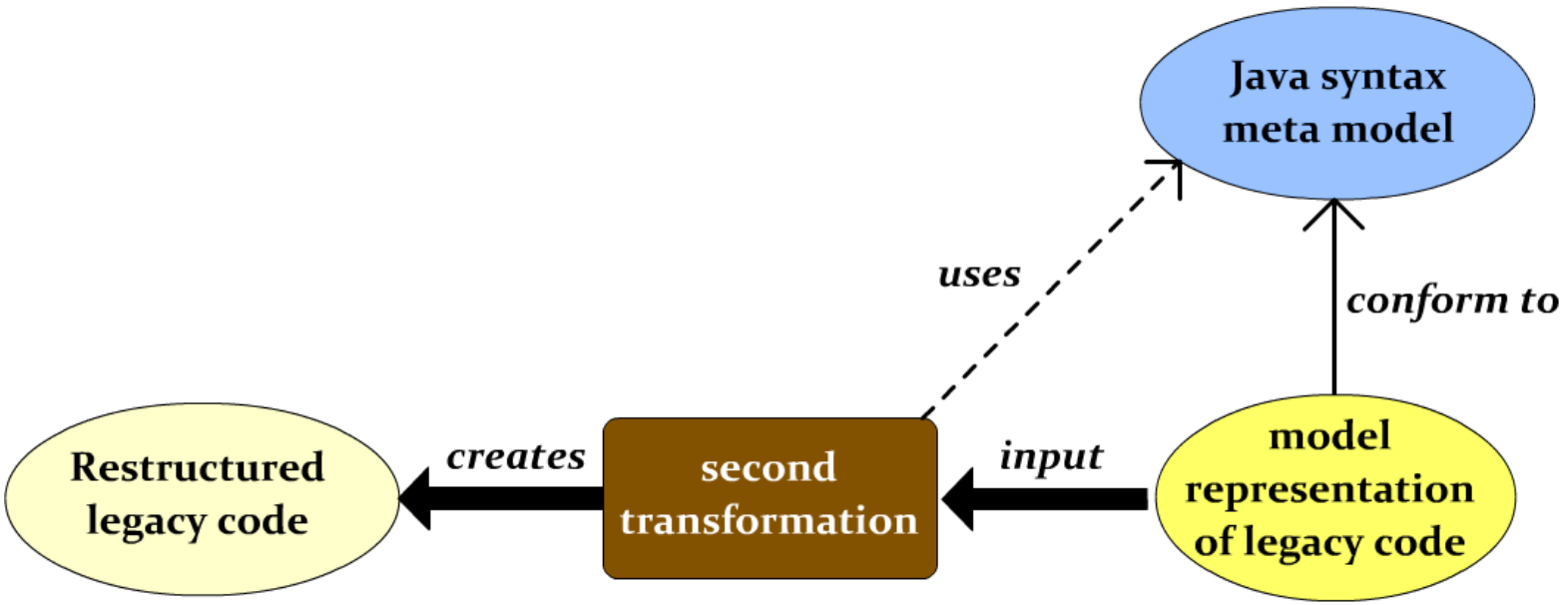
Note that the first transformation step does not raise the abstraction level. The gained model holds exactly the same information as the existing code base does. Hence we call this transformation an "information mapping".

First Transformation — Using Java as an Example



- **input**: Abstract Syntax Tree (AST)
- **creates**: XMI model

Second Transformation — Using Java as an Example



- **input**: XMI model
- **creates**: Java code

Second Transformation — A Closer Look

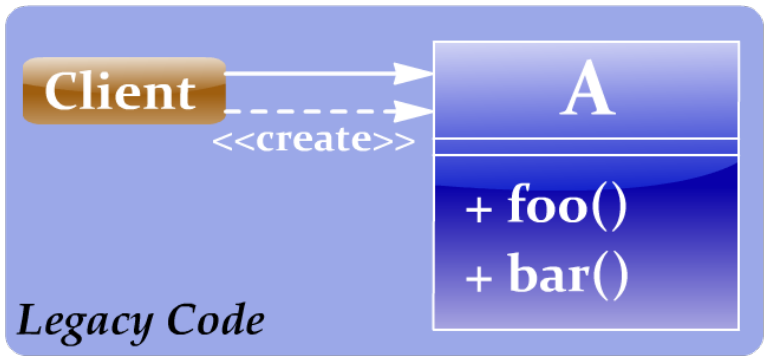
How is the legacy code actually restructured?

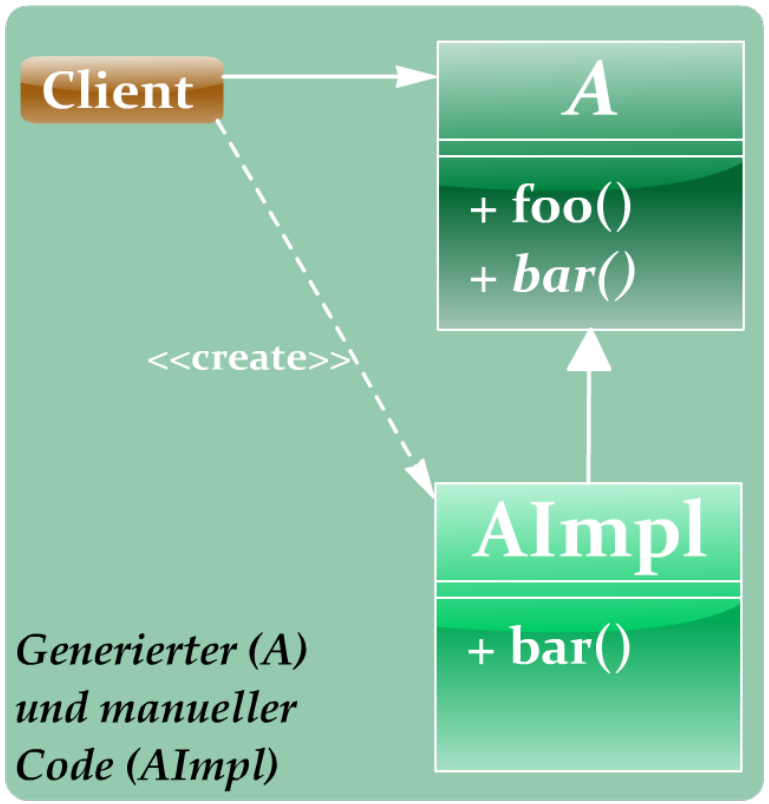
- forward engineering knows several patterns or best practices to combine generated and h/c code
  - protected regions
  - inheritance mechanism
  - ...
- for reverse engineering they need to be reconsidered
- let's take a look at the inheritance approach

Inheritance Approach

- Idea: Generate (abstract) base classes which can be extended by subclasses if necessary
- Using this for restructuring would have the following disadvantages:
  a) existing inheritance hierachy would be corrupted
  b) either the creation– or the usage–interface of a class would change
  c) between every generated base class and existing subclass there would only be 1:1 dependencies

Inheritance Approach

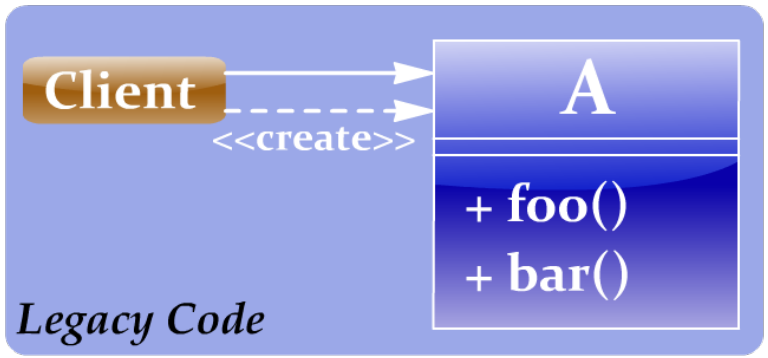


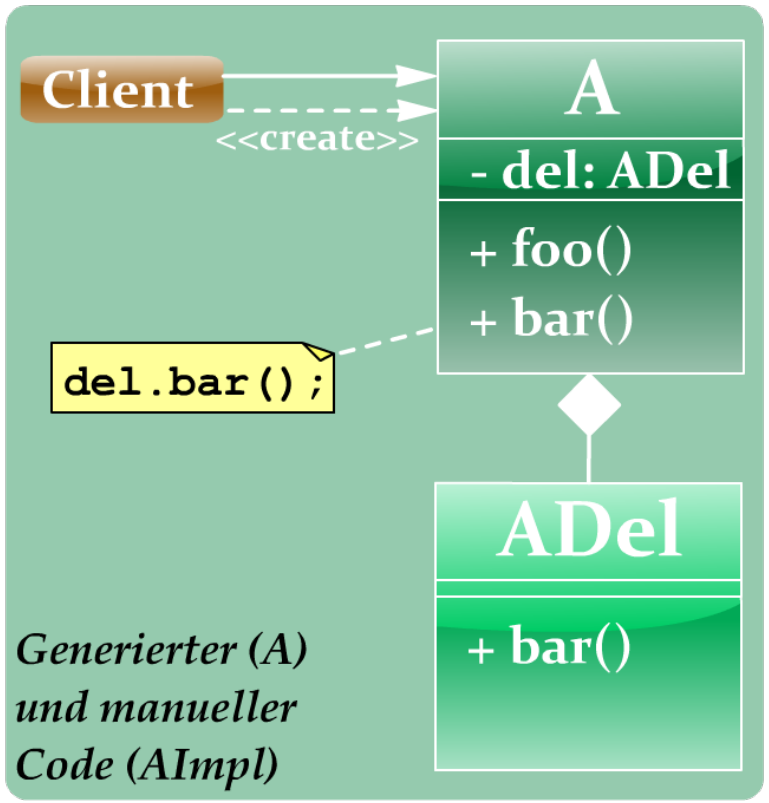


Delegation Approach

- Idea: Generate classes that delegate client calls if necessary to manually implemented classes
- Several advantages:
  a) possible to steer the handing back and forth of execution control in a very fine grained manner
  b) results of manually written code can be validated at runtime by embedding appropriate reactions in the generated code
  c) the existing inheritance hierarchy does not get corrupted
  d) creation and usage interface stay the same ($\Rightarrow$ Less manual adjustments necessary)

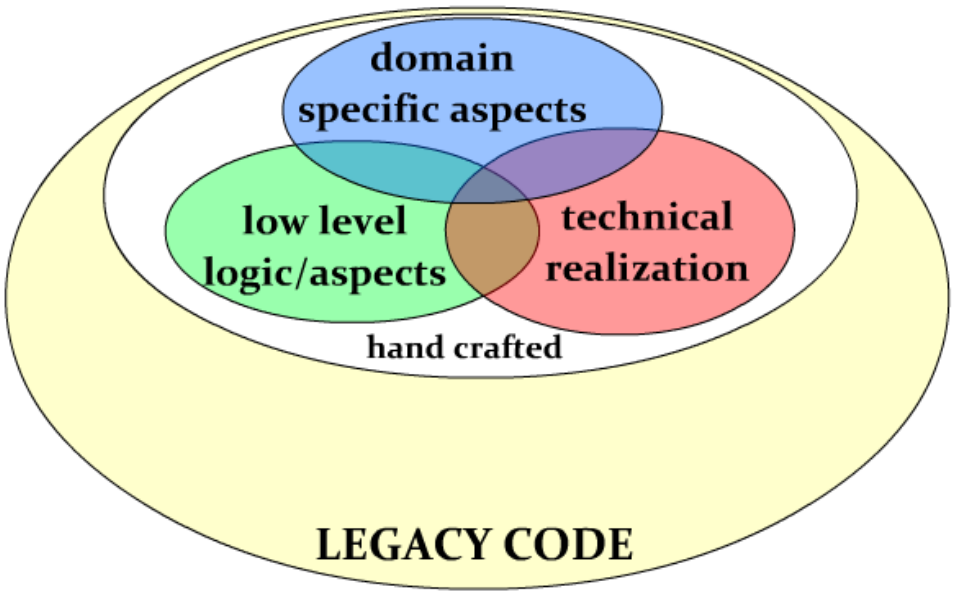
Delegation Approach



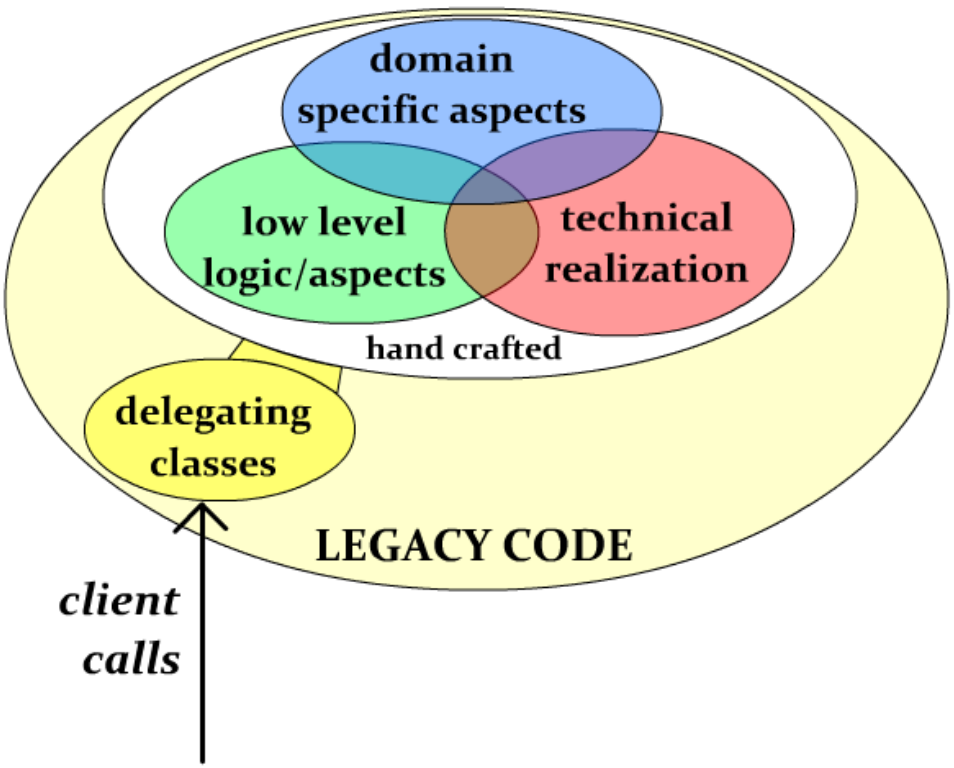


Overview

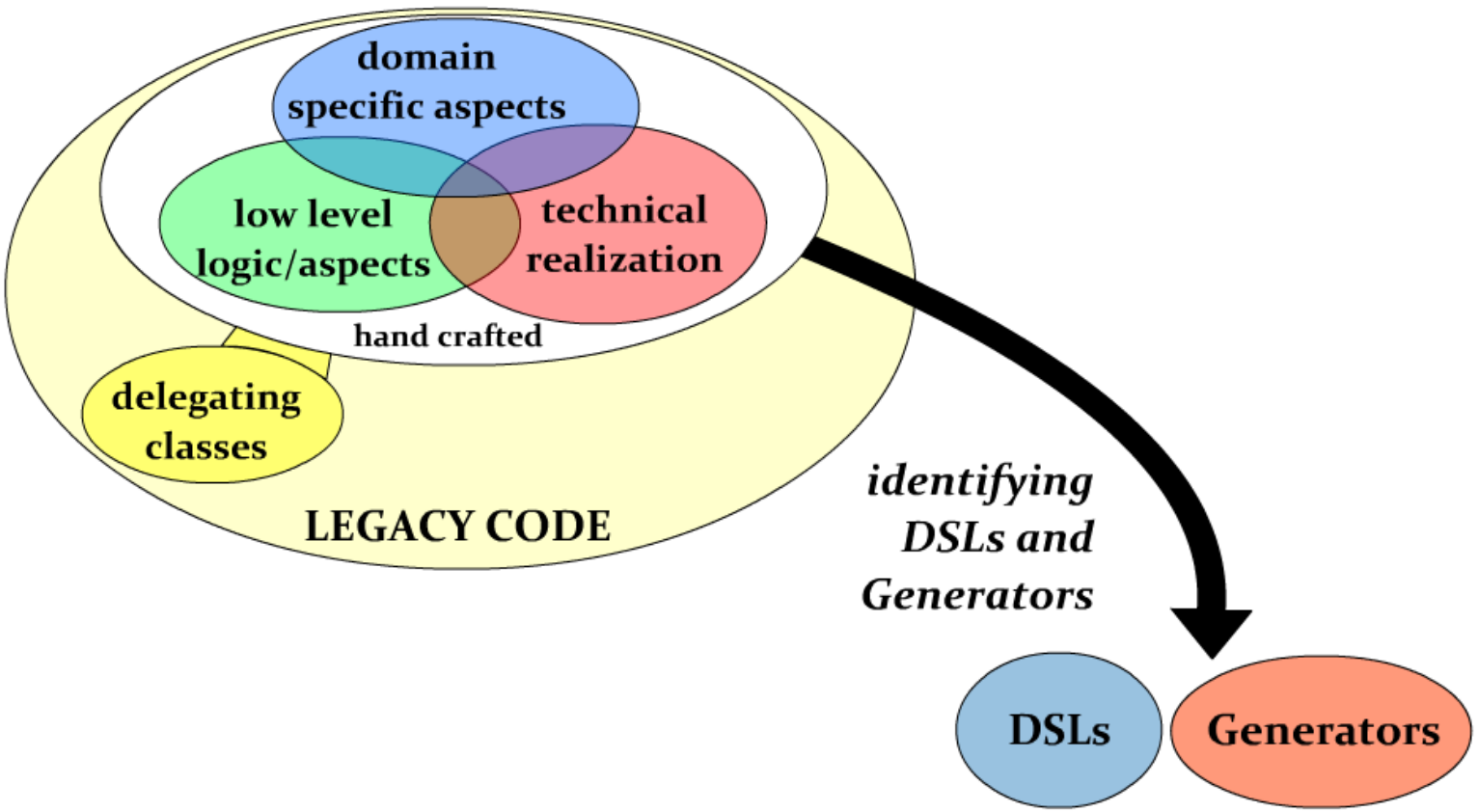
Restructuring and Further Iterations



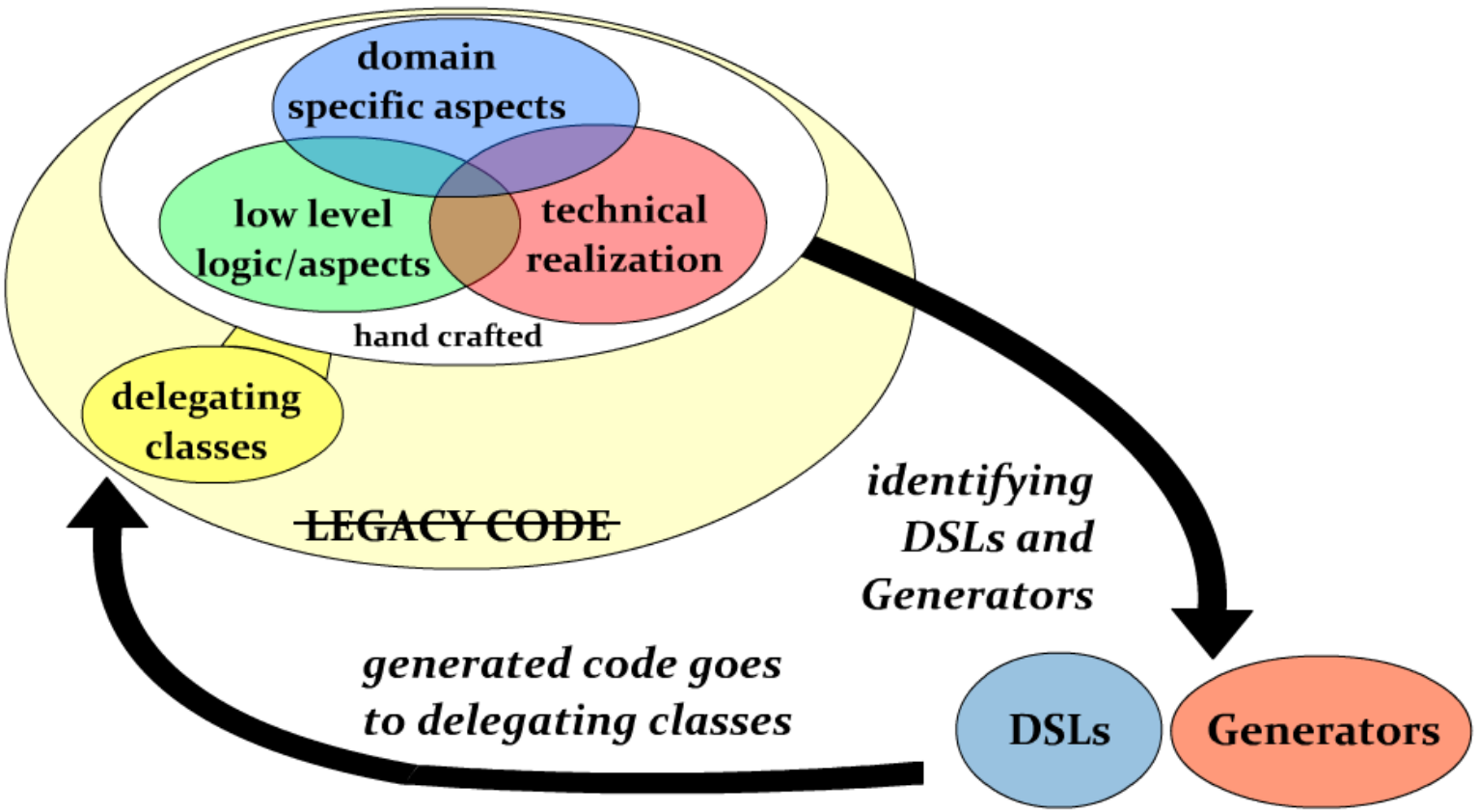
Restructuring and Further Iterations



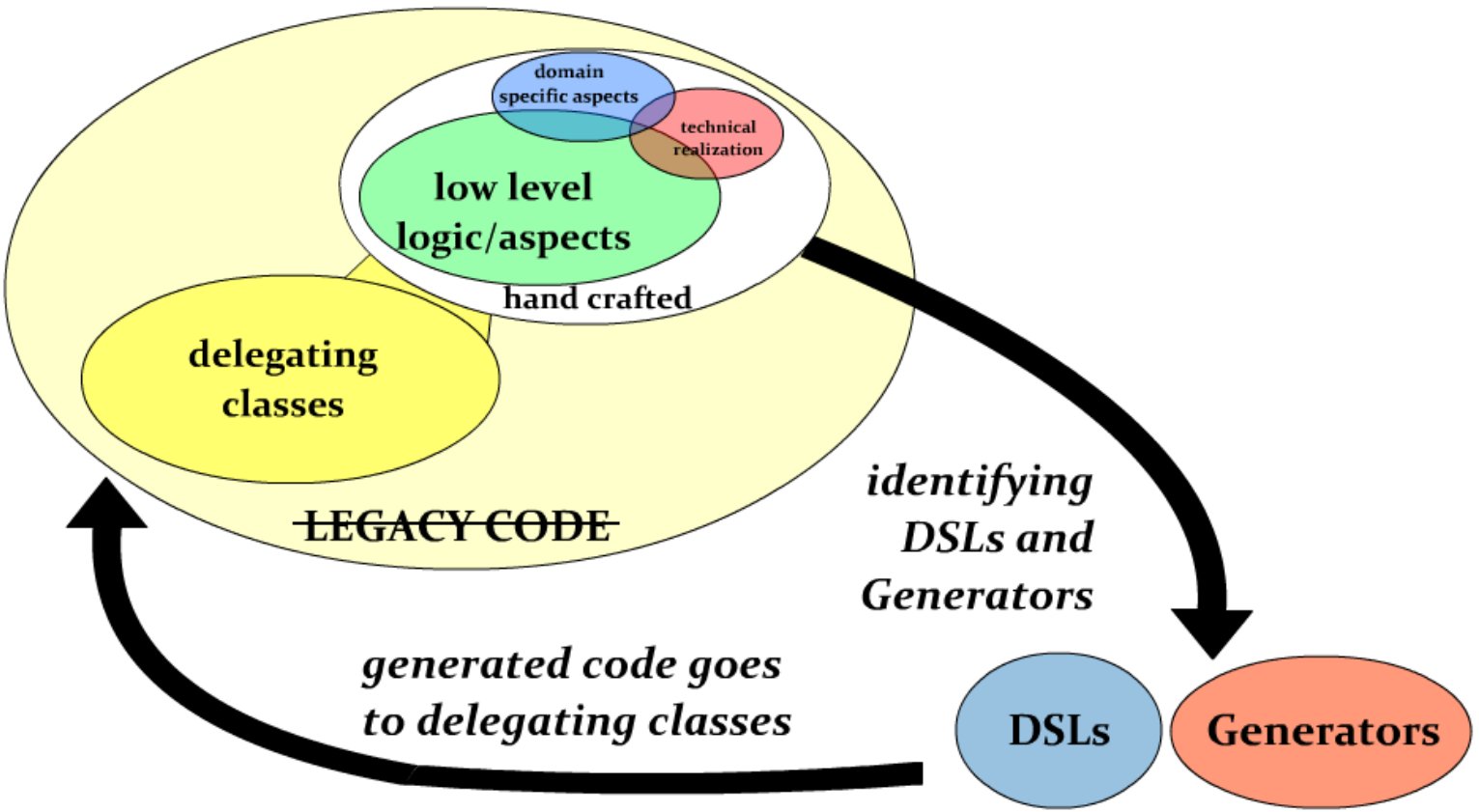
Restructuring and Further Iterations



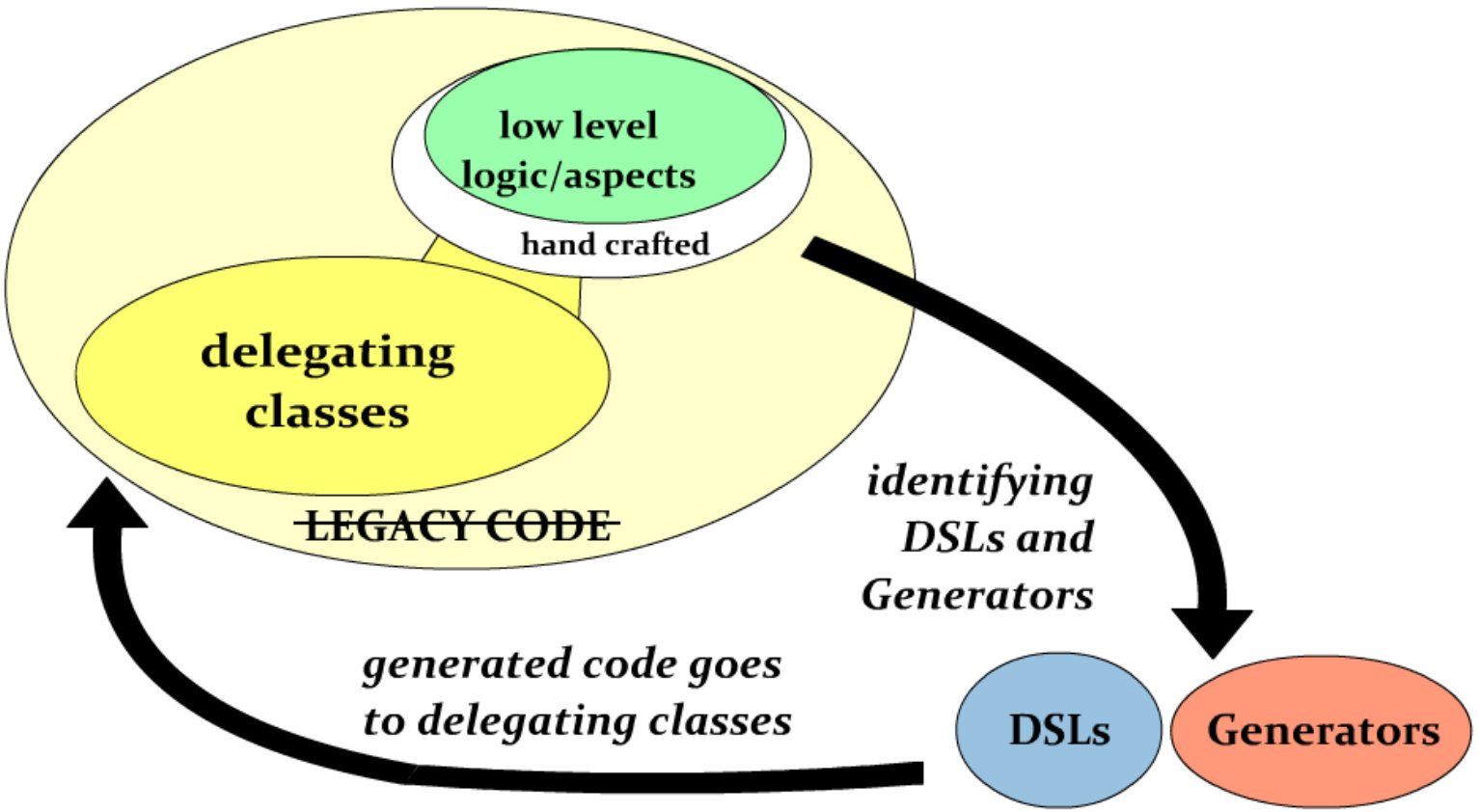
Restructuring and Further Iterations



Restructuring and Further Iterations



Restructuring and Further Iterations



Outlook

- extending / validating Java language meta model
- validating delegation approach using industrial reference projects
- enhance restructuring to enable refactoring the legacy code base
- carry over delegation approach to other languages
- tool support of domain identifiying process