Designing a Domain Specific Language and Its Transformer for Back-end Development of Information Management Systems

Yige Wen (Master of Information Technology, University of Melbourne),
Hidehiko Masuhara (Department of Mathematical and Computing Science, Tokyo Institute of Technology)
Github: https://github.com/HikaruGE/Information-System-Management-DSL

Background

Information management system: is commonly used in libraries, universities and companies. It provides users with functions of creating, retrieving, updating and deleting the records. Besides, its defined transaction logic could be able to constraint users' behaviors and link different User Interface.

Domain Specific Language (DSL): comparing to general purpose language, Java, python and C++, its usage scenario is single and specific, which is designed for a particular domain, e.g. HTML for web and SQL for data-base. The objective of DSL is to help programmers get more productive by the linguistic abstraction of some certain domain.

Problems

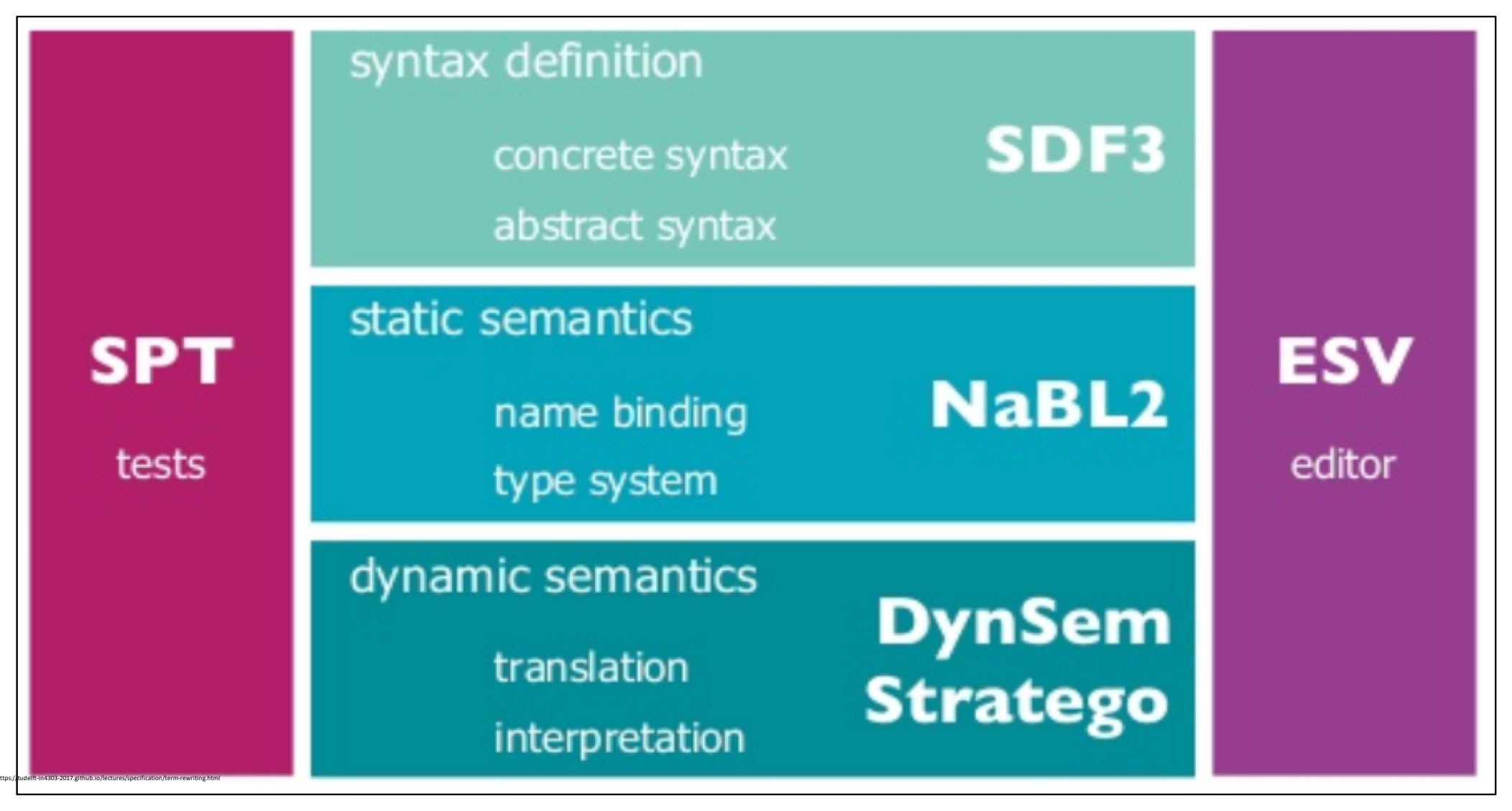
- A lot of structurally similar or repeated code, which could be easily leading to mistakes;
- In the software development process, the conflict between the development team and the customers without programming experience.

Goal

Designing a domain specific language of the information management system to achieve the followings:

- Get rid of the quantity of handwriting code;
- Easily understood semantic;
- Faster release.

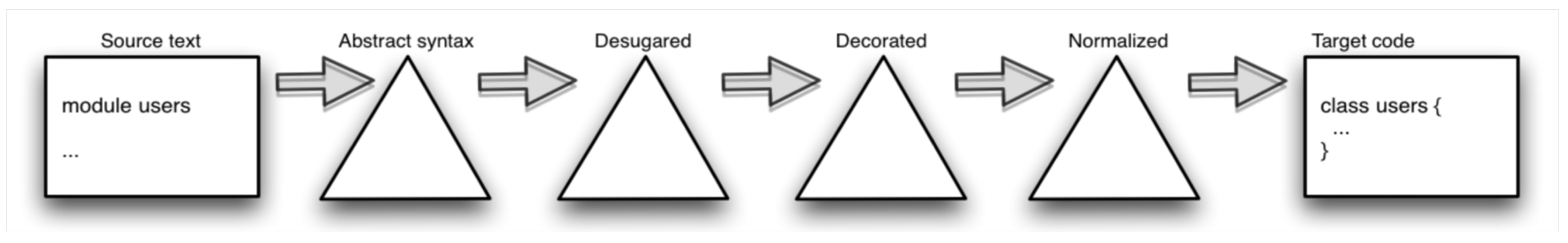
Approach: Programming language workbench,

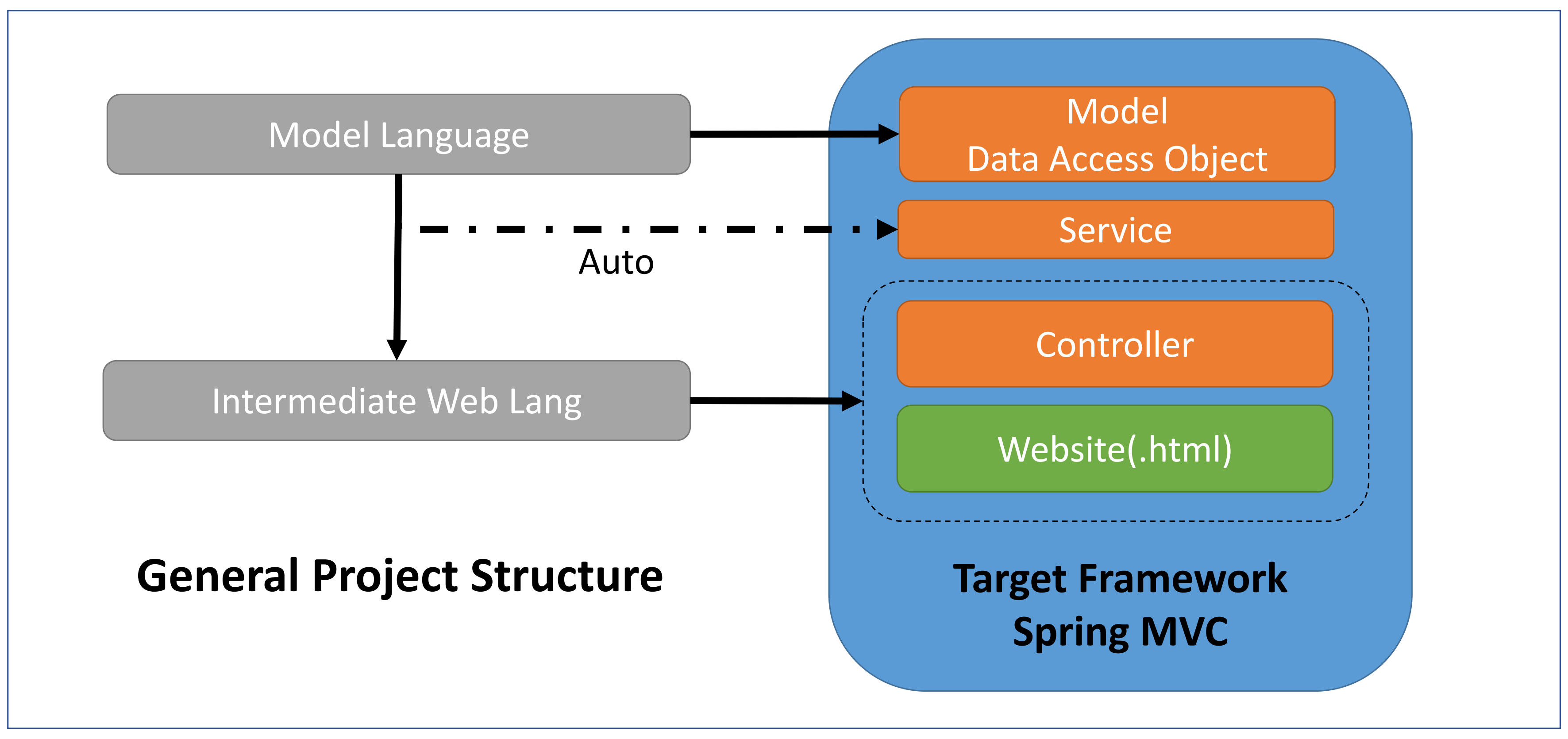


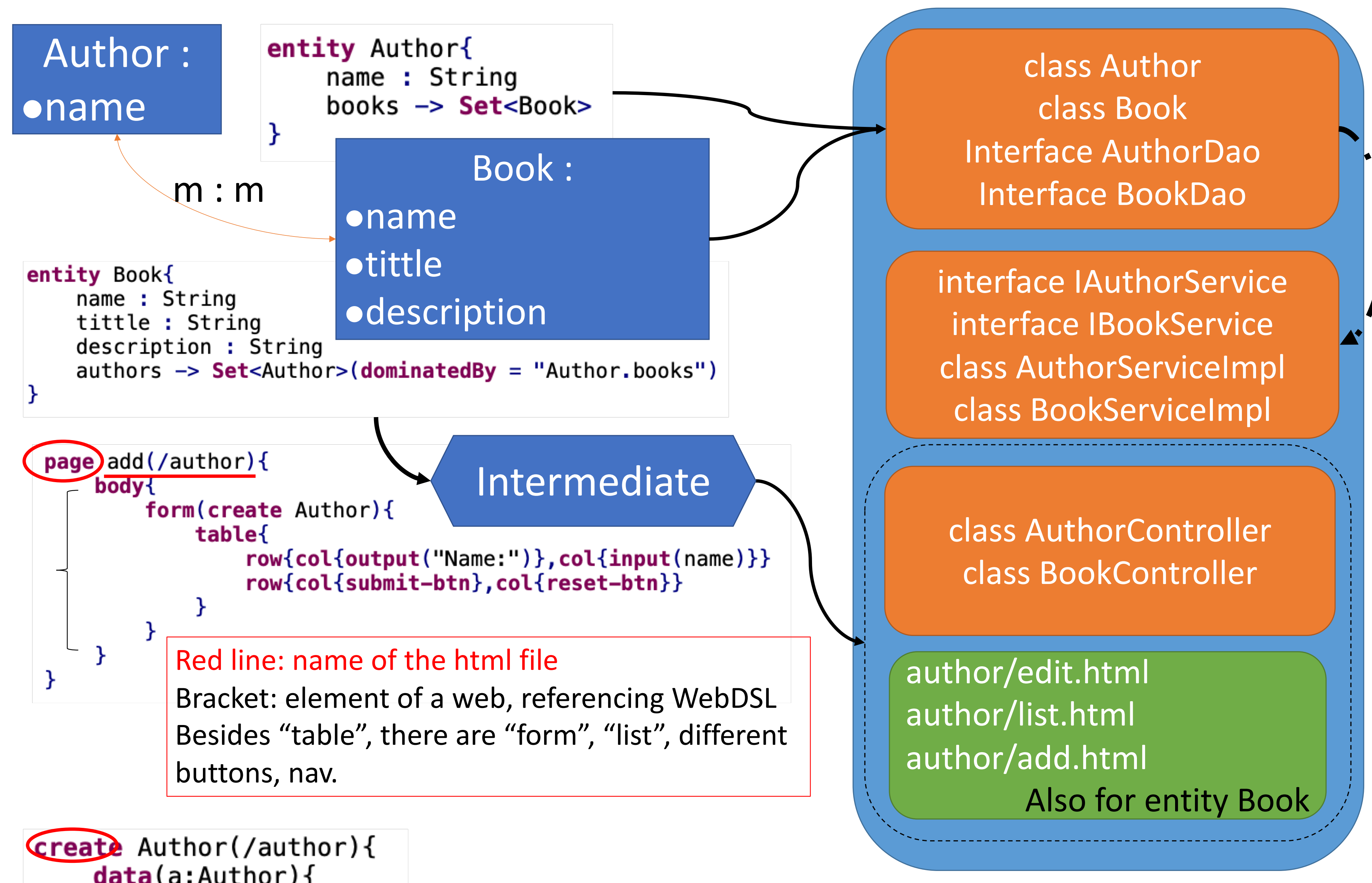
Example:

■SDF3 (ATerms)
3+2*7 is Add(3,Mul(2,7))

■Stratego (Re-write)
Add(x,y) -> z
where z eq x+y







```
entity Author{
     name : String
     books -> Set<Book>
}
```

```
entity Book{
    name : String
    tittle : String
    description : String
    authors -> Set<Author>(dominatedBy = "Author.books")
}
```

```
page add(/author){
    body{
        form(create Author){
            table{
                row{col{output("Name:")},col{input(name)}}
                row{col{submit-btn},col{reset-btn}}
            }
        }
    }
}
```

Red line: name of the html file
Bracket: element of a web, referencing WebDSL
Besides "table", there are "form", "list", different buttons, nav.

```
create Author(/author){
    data(a:Author){
    Author.save(a)
    }
    goto(/studentList)
}
```

Code for Author's service's interface

```
public interface IAuthorService {
   List<Author> findAll(); //R
   void save(Author author); //C,U
   void deleteById(Long id); //D
   Author findStudentById(Long id); //R
}
```

If only CRUD, user don't need define other service

The author of WebDSL used a different web framework though, he used sentence "derive CRUD <entity>" to let users can quickly invoke a lot of automatically generated method without handwriting them.

Design Thought:

When rendering pages, it will use data in controller.

When sending requests, the restful style is chosen, distinguishing requests by the verb in methods, no longer in path, which is more uniformed and well-defined.

	URL	Method	Html file
Page: All Entity	/[entity]Lst	GET	/[entity]/list.html
Page: Add Entity	/[entity]	GET	/[entity]/add.html
Page: Edit Entity	/[entity]/{id}	GET	/[entity]/edit.html
Operation: create	/[entity]	POST	
Operation: update	/[entity]	PUT	
Operation: delete	/[entity]/{id}	DELETE	

Table: URL design of restful style

Evaluation:

Metric: Lines of Code (LoC)

	DSL	Spring MVC	~%
Entity Author	4	27	14.8
Entity Book	6	37	16.2

Conclusion:

Using DSL can really reduce the number of lines of code, avoid unnecessary duplication of code, and only need to pay attention to the data stream to complete server development, even the web page jump is not need to be considered. But this is only for simple situation, suppose if the system is only open for the authenticated users, login and register functions are required and each user may have its own role to implement access control.

Future Works:

In this project, only part of the target system is mapped to the designed DSL, other functions, like authority management, paging query and so on. So next step is to integrate new functions as more as possible.
By learning this language workbench, thanks to its powerful rewrite mechanism, translating programming languages with similar compilation methods can be promising.