



A DSL to create REST services

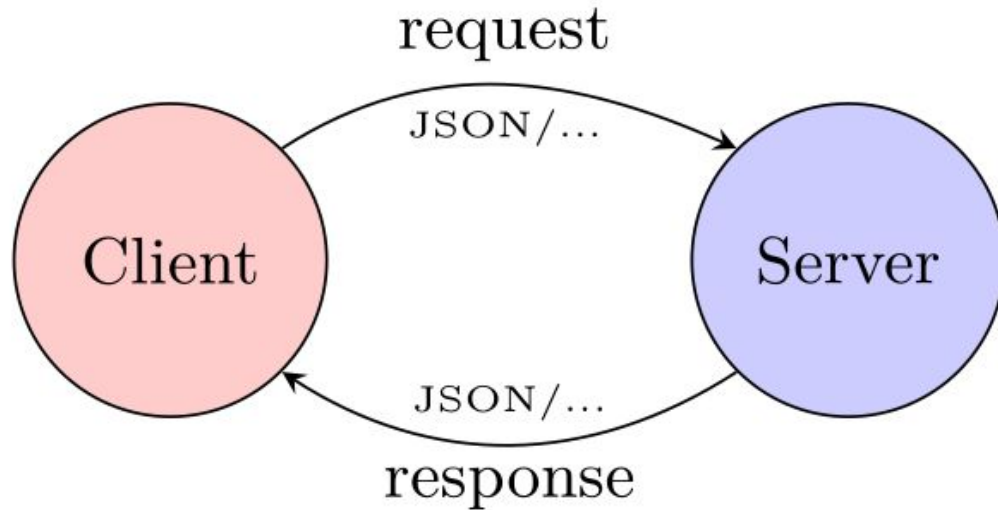
Jorge Blázquez, Enrique de la Calle



Index

- Introduction
 - REST APIs
 - DSLs
- Metamodel
- XText DSL
 - Validations
 - Suggestions
- Transformations
 - DSL -> Instance
 - XMI -> Ecore
 - M2T
- Demo
- Conclusion

REST APIs





REST APIs

- POST: es el equivalente a la operación CREATE, con el que podemos crear un nuevo recurso.
- GET: es el equivalente a la operación READ, con el que podemos leer el contenido de un recurso ya existente.
- PUT: es el equivalente a la operación UPDATE, con el que podemos modificar un recurso que ya existe.
- DELETE: es el equivalente a la operación DELETE, con el que podemos eliminar un recurso.



DSLs

```
struct Page {  
    field num;  
}  
  
struct Book {  
    field author;  
    field title;  
    field pageId;  
}  
  
post "book" {  
    param author;  
    param title;  
    param pageId;  
    random id;  
    create Book(author: author, title: title, pageId: pageId) with id;  
    status 201;  
    return "{id}";  
}  
  
get "books" id {  
    read Book(author, title, pageId) with id if fail {  
        status 404;  
        return "Book_not_found"  
    };  
    return "Title:_{title},_Author:_{author}";  
}  
  
get "books" id "name" {  
    read Book(author, title, pageId) with id if fail {  
        status 404;  
        return "Book_not_found"  
    };  
    return title;  
}  
  
put "books" id {  
    param newAuthor;  
    param newTitle;  
    update Book(author: newAuthor, title: newTitle) with id if fail {  
        status 404;  
        return "Book_not_found";  
    };  
    return "Book_with_id_{id}'_updated";  
}
```

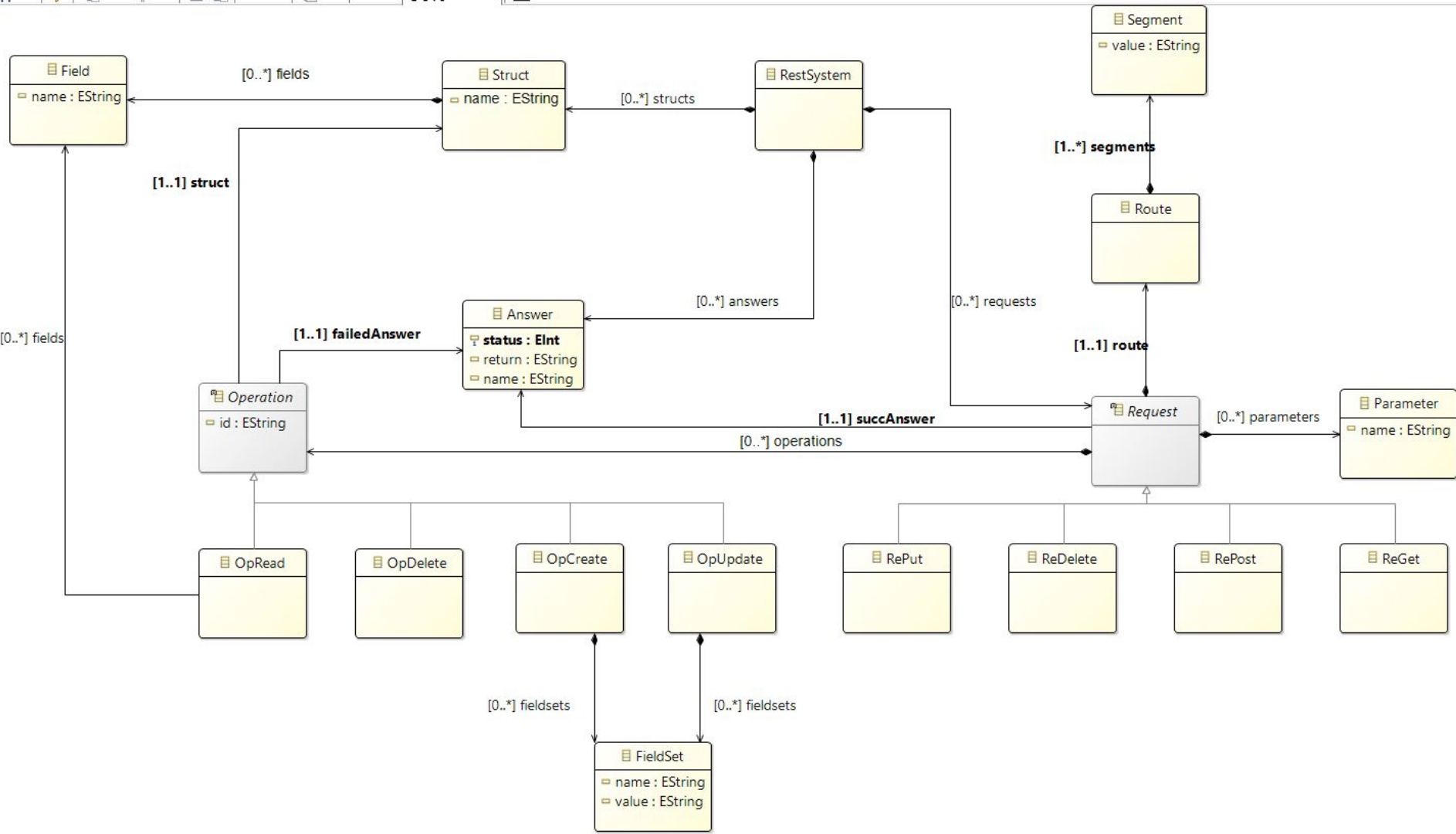


Designing the model

A tradeoff between flexibility and complexity.

We could just remove operations and just forcing the request to certain behaviour, but what if our user wants to make multiple reads in the same request?

We also give the user the flexibility to use variables for messages and operation





XText

Our DSL has been implemented in XText, which allows us to use a custom editor.

We can code now our own ".rest" files!



XText Validation:

What if we want Structs to begin with capital letters to comply with Java classes? Or making sure the fields accessed are correct?

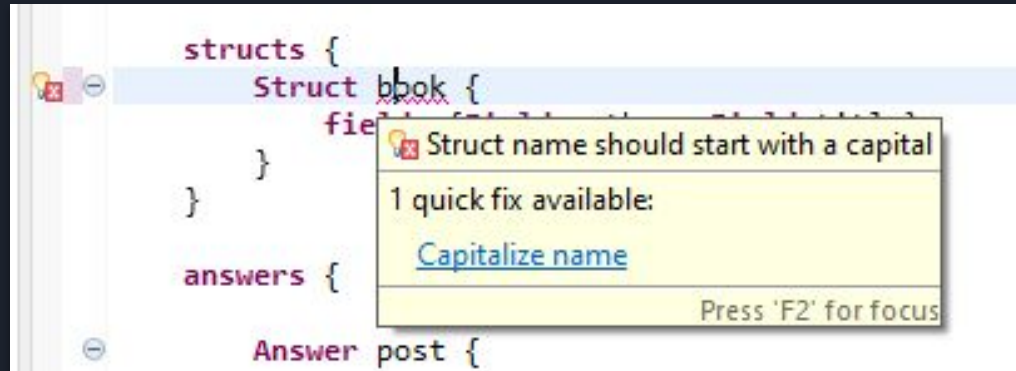
We don't want to use OCL...

Solution: custom XText validators

```
@Check
public void structWithCaptital(Struct struct) {
    if (!Character.isUpperCase(struct.getName().charAt(0))) {
        error("Struct name should start with a capital", RestModelPackage.Literals.STRUCT__NAME, "invalidName");
    }
}
```

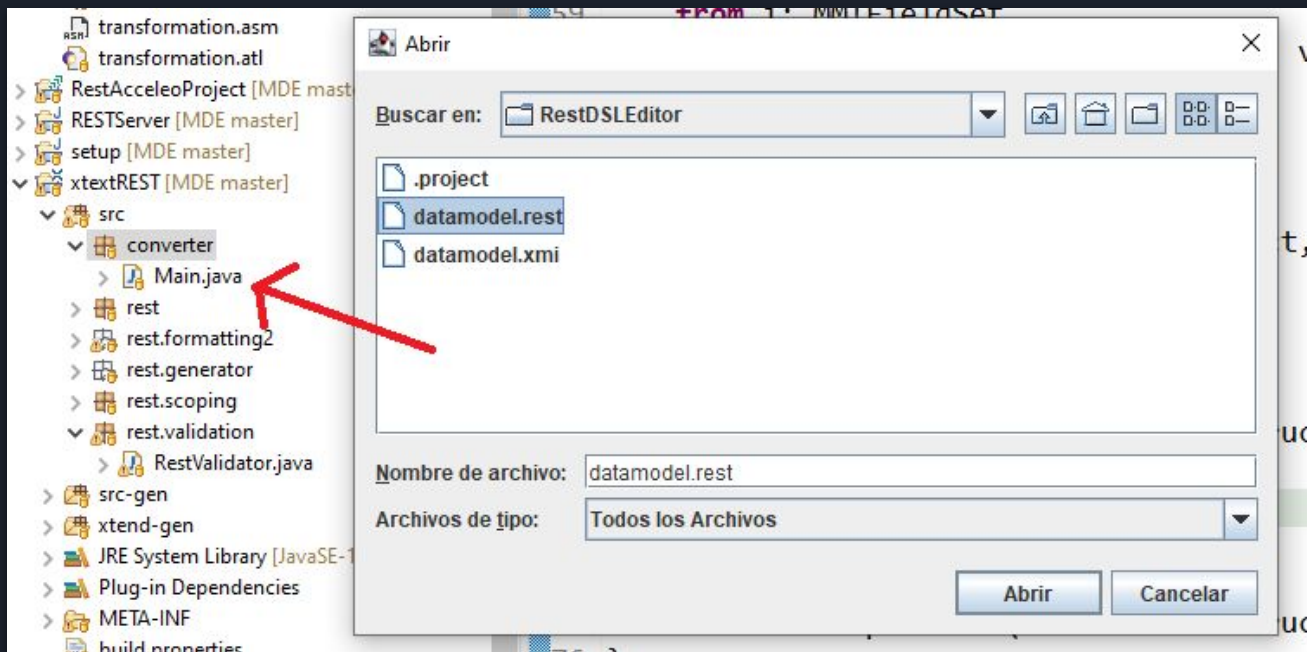
XText Suggestions:

We have created a custom suggestion to fix capitalization



.rest -> .XMI

A quick script in the "xtextREST" project has been coded to be able to convert our DSL files into dynamic instances of the metamodel.





M2M .XMI -> .ecore

We also coded a M2M, which is able to convert those dynamic instances into .ecore

This M2M has been implemented into ATL, by just mapping the same metamodel to itself

Now we can see the datamodel as an ecore!

```
73 rule OpCreate{  
74   from i: MM!OpCreate  
75   to o: MM2!OpCreate (struct <- i.struct, failedAnswer <- i.failedAnswer, fieldsets <- i.fieldsets, id <- i.id)  
76 }
```



M2T

We already have a dynamic instance from the .rest file, now we want to generate an entire REST server.

An empty project ("RESTServer") with the necessary dependencies (Spark library) is setup.

For each struct, a new java class file is created with the name of that struct. The fields will then generate the local variables with their corresponding getter and setter functions.

For each request, a new call to the Spark library is written, according to the corresponding request type.

```
[template public generateStructFile(struct : Struct)]

[file (struct.name.concat('.java'), false, 'UTF-8')]
public class [struct.name/] {


    [for (field : Field | struct.fields)]
    private String [field.name/];
    [/for]

    public [struct.name/]( [for (field : Field | struct.fields) separator (' ', ')]String [field.name/][[/for]]){
        [for (field : Field | struct.fields)]
        this.[field.name/] = [field.name/];
        [/for]
    }


    [for (field : Field | struct.fields)]
    public String get[field.name.toUpperFirst()/](){ return [field.name/];};
    [/for]

    [for (field : Field | struct.fields)]
    public void set[field.name.toUpperFirst()/](String [field.name/]){ this.[field.name/] = [field.name/];};
    [/for]
}
[/file]


[/template]
```



```
public class Book {  
  
    private String author;  
    private String title;  
  
    public Book(String author, String title){  
        this.author = author;  
        this.title = title;  
    }  
  
    public String getAuthor(){ return author;};  
    public String getTitle(){ return title;};  
  
    public void setAuthor(String author){ this.author = author;};  
    public void setTitle(String title){ this.title = title;};  
}
```



```
8 [template public generateReadOp(anOp : OpRead)]
9 [sName(anOp)/] [oName(anOp)/] = [lName(anOp)/].get([anOp.id/]);
10 if ([oName(anOp)/] != null) {
11     [for (field : Field | anOp.oclAsType(OpRead).fields)]
12     String [field.name/] = [oName(anOp)/].get[field.name.toString().toUpperFirst()/]();
13     [/for]
14     [succAnswer(anOp)/]
15 } else {
16     [failAnswer(anOp)/]
17 }
18 [/template]
19 |
```



```
Book book = books.get(id);
if (book != null) {
    String author = book.getAuthor();
    String title = book.getTitle();
    response.status(200);
    return "Title: " + title + ", Author: " + author;
} else {
    response.status(404);
    return "error";
}
```




DEMO





Conclusions

A metamodel for describing REST servers has been designed.

A DSL from the metamodel has been coded in XText, with some additional restrictions.

An small script has been added which transforms the DSL files to .xmi.

A transformation in ATL has been coded, which converts the .xmi into .ecore.

A M2T has been implemented which turns the files in the DSL into Java applications.

One of this applications has been tested from an example.