

# A model-driven approach for REST compliant services



University of Stuttgart  
Universitätsstr. 38  
70569 Stuttgart  
Germany

Florian Haupt  
Institute of Architecture of Application Systems  
[florian.haupt@iaas.uni-stuttgart.de](mailto:florian.haupt@iaas.uni-stuttgart.de)

Phone +49-711-685 88205  
Fax +49-711-685 88472



# Agenda

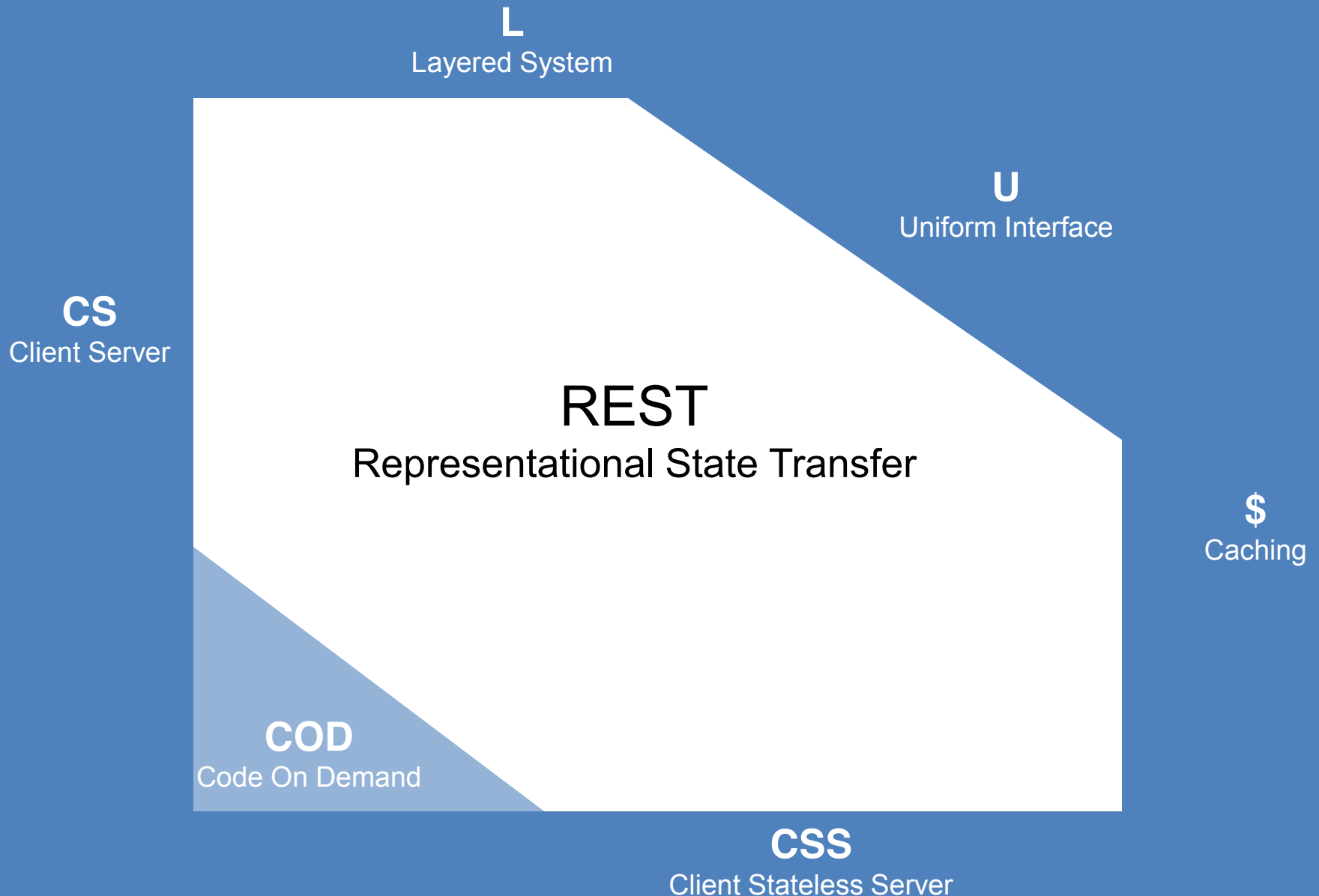
---

- REST Introduction
- REST and the WWW
- Layered Meta-Model
- Prototype
- Formalization
- Conclusion

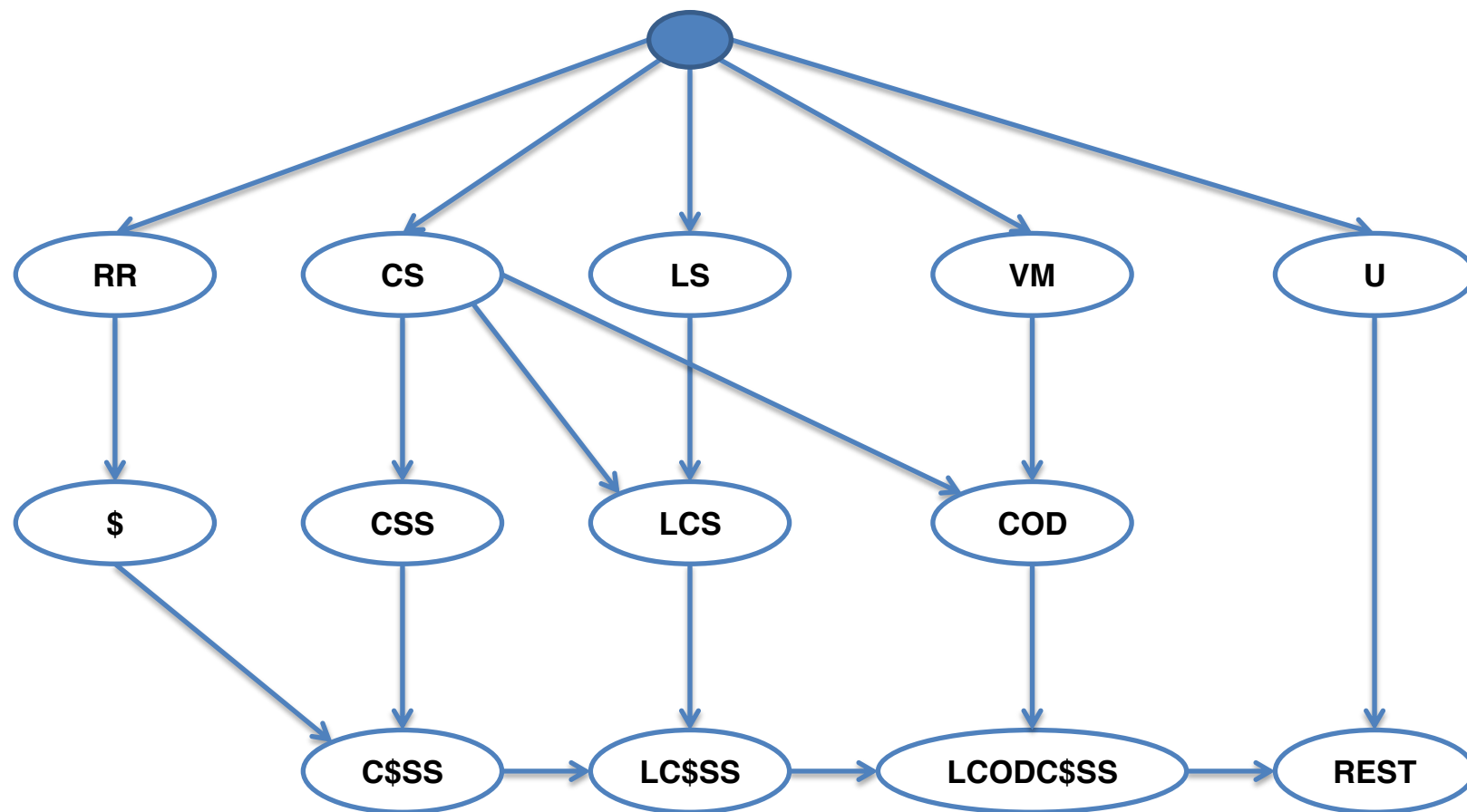
# Definition

- REST is an “architectural style for distributed hypermedia systems” [2]
  - The WWW is a distributed hypermedia system
- REST is defined as a set of architectural constraints

# REST = LCODC\$SS + U



# REST style derivation tree



# Uniform Interface

- Identification of resources
- Manipulation of resources through representation
- Self-descriptive messages
- Hypermedia as the engine of application state
  - aka HATEOAS

# Classification of architectural styles

Style	Derivation	Net Perform.	UP Perform.	Efficiency	Scalability	Simplicity	Evolvability	Extensibility	Customiz.	Configur.	Reusability	Visibility	Portability	Reliability
PF			±			+	+	+		+	+			
UPF	PF	-	±			++	+	+		++	++	+		
RR			++		+									+
\$	RR		+	+	+	+								
CS					+	+	+							
LS			-		+		+				+		+	
LCS	CS+LS		-		++	+	++				+		+	
CSS	CS	-			++	+	+					+		+
C\$\$\$	CSS+\$	-	+	+	++	+	+					+		+
LC\$\$\$	LCS+C\$\$\$	-	±	+	+++	++	++				+	+	+	+
RS	CS			+	-	+	+					-		
RDA	CS			+	-	-						+		-
VM						±		+				-	+	
REV	CS+VM			+	-	±		+	+			-	+	-
COD	CS+VM		+	+	+	±		+		+		-		
LCODC\$\$\$	LC\$\$\$+COD	-	++	++	+4+	±±	++	+		+	+	±	+	+
MA	REV+COD		+	++		±		++	+	+		-	+	
EBI				+	--	±	+	+		+	+	-		-
C2	EBI+LCS		-	+		+	++	+		+	++	±	+	±
DO	CS+CS	-		+			+	+		+	+	-		-
BDO	DO+LCS	-	-				++	+		+	++	-	+	

# The History of REST

- 1991
  - HTTP 0.9
  - First version by Tim Berners-Lee
- 1994
  - „Universal Resource Identifiers in WWW“
    - RFC 1630
  - „Uniform Resource Locators (URL)“
    - RFC 1738
- 1996
  - HTTP 1.0
    - RFC 1945
- 1999
  - HTTP 1.1
    - RFC 2616
- 2000
  - “Architectural styles and the design of network-based software architectures”
  - Dissertation by Roy Fielding
  - First notion of the term “REST”

# Motivation

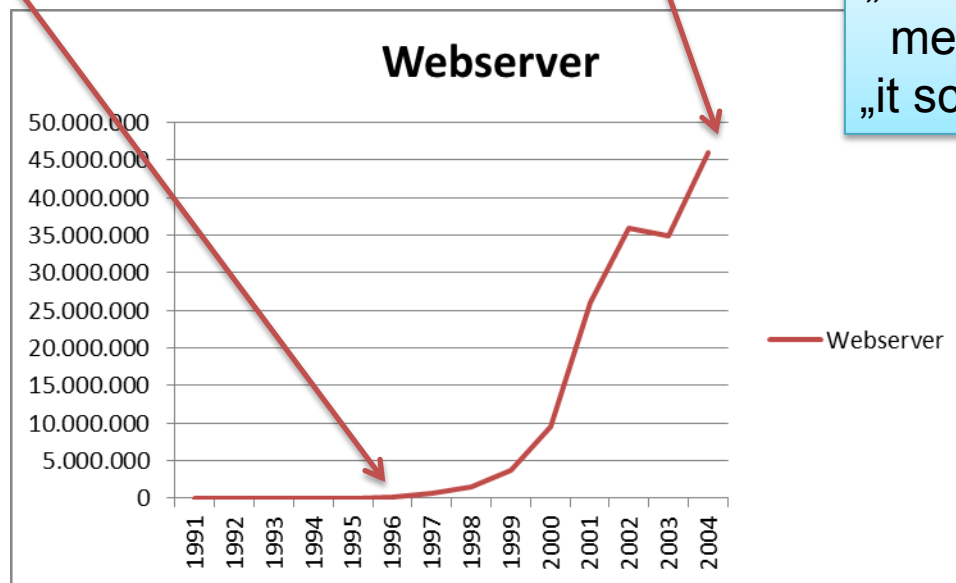
- “Since 1994, the REST architectural style has been used to guide the design and development of the architecture for the modern Web.” [2]
- “... the motivation for developing REST was to create an architectural model for how the Web should work, such that it could serve as the guiding framework for the Web protocol standards. REST has been applied to describe the desired Web architecture, help identify existing problems, compare alternative solutions, and ensure that protocol extensions would not violate the core constraints that make the Web successful. This work was done as part of the Internet Engineering Taskforce (IETF) and World Wide Web Consortium (W3C) efforts to define the architectural standards for the Web: HTTP, URI, and HTML.” [2]

# Success

The basics have been designed here  
(100.000 Webservers)

It worked here  
(46.000.000+ Webservers)

And still works today  
(02/2012 Apache had a market  
share of 64,92% with 397.867.089  
servers)



„it works“  
means  
„it scales“

Numbers are taken from:

- <http://www.w3.org/2005/01/timelines/description>
- <http://news.netcraft.com/archives/category/web-server-survey/>

# One reason for success

- “REST is software design on the scale of decades: every detail is intended to promote software longevity and independent evolution.” [2]

# It looks easy, but it is hard to achieve

- “However, I think most people just make the mistake that it should be simple to design simple things. In reality, the effort required to design something is inversely proportional to the simplicity of the result. As architectural styles go, REST is very simple.” [1]









# REST and the WWW



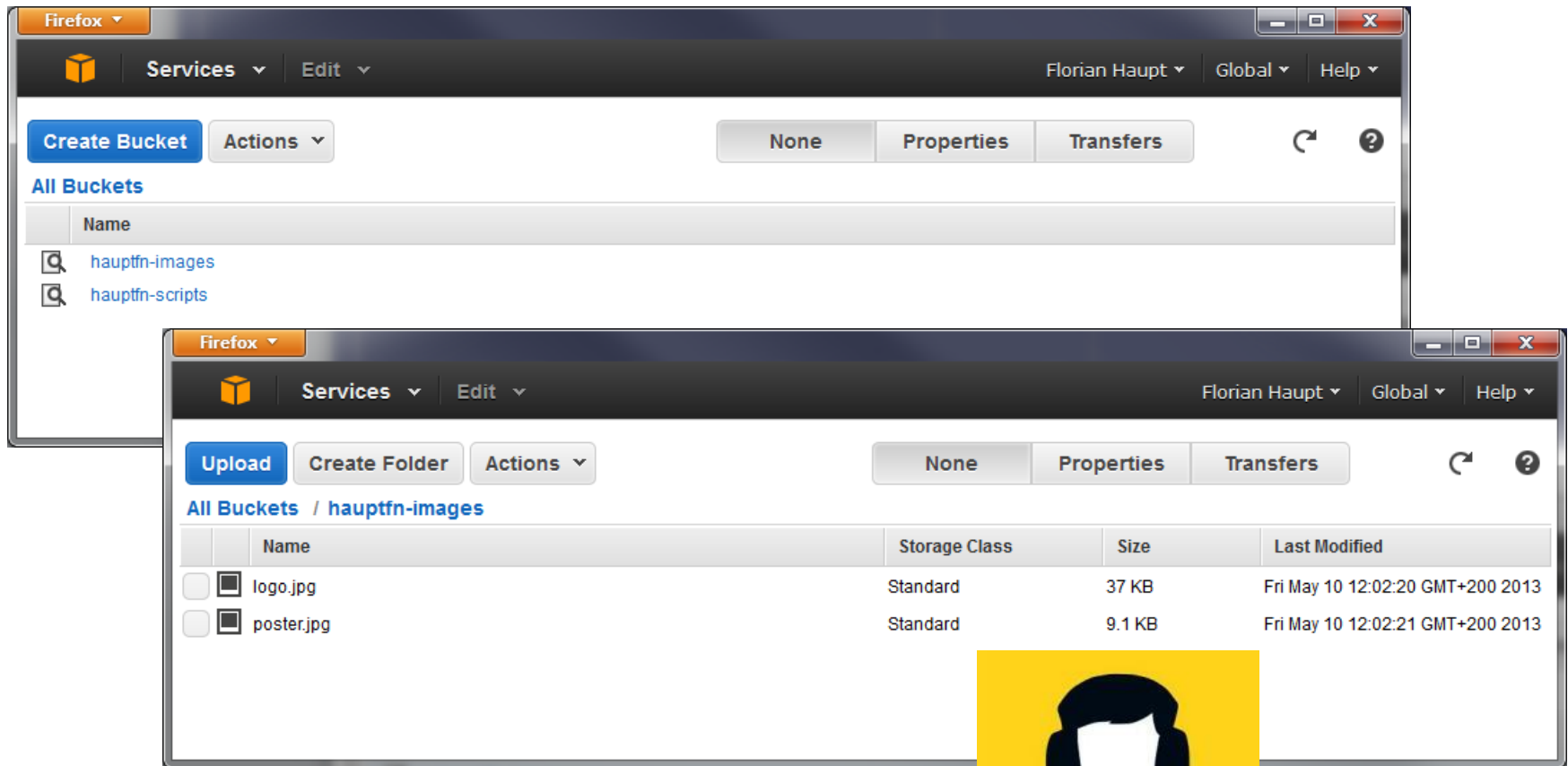
# REST and the WWW – Introduction

- Representational State Transfer (REST)
  - An architectural style for distributed hypermedia systems
  - Defined as a set of constraints each REST architecture has to comply with
- World Wide Web (WWW)
  - The best known and biggest system following the REST architectural style
  - The WWW is based on several standards
    - HTTP, URI, MIME, HTML
- How are the REST constraints fulfilled in the WWW?

# REST and the WWW – Fulfilling the constraints

REST constraint	Fulfilled by	
Layered Client Server	HTTP / default	
Cache	HTTP	
Stateless Server	Developer	
Uniform Interface	HTTP + Developer	
Identification	HTTP / URI	
Manipulation through Representations	HTTP	
Self Descriptive Messages	HTTP	
HATEOAS	Developer (User?)	

# How to retrieve an object from an S3 bucket (1)



# How to retrieve an object from an S3 bucket (2)

GET s3.amazonaws.com

```
<ListAllMyBucketsResult>
  <Buckets>
    <Bucket>
      <Name>hauptfn-images</Name>
    </Bucket>
    <Bucket>
      <Name>hauptfn-scripts</Name>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

GET hauptfn-images.s3.amazonaws.com

```
<ListBucketResult>
  <Name>Images</Name>
  <Contents>
    <Key>poster.jpg</Key>
  </Contents>
  <Contents>
    <Key>logo.png</Key>
  </Contents>
</ListBucketResult>
```

GET hauptfn-images.s3.amazonaws.com/poster.jpg

```
<Key>/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAA
gGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8U
HRofHH0aHBwgJC4nICIsIxwcKDcpLDAxND
Q0Hyc5PTgyPC4zNDL/2wBDAAQkJCQwLDBgN
DRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMj
IyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy
MjIyMjL/wAARCAEsANUDASIAAhEBAxEB/8
QAHAABAAICAwEAAAAAAAAAAAAAAAAAEHBQYD
BAgC/8QATxAAAQMDAQUCQYHDwMFAAAAAAQ
ACAwQFEQYHEhMhMUFRFBUIMmFxxgZGhFkJS
sCHRI1Vic3STsggXJDM1NjdTcPKUosLh8E
NUgmNkhLPi/8QAGwEBAAIDAQEAAAAAAAAA...
```

GET {BucketName}.s3.amazonaws.com/{ObjectKey}



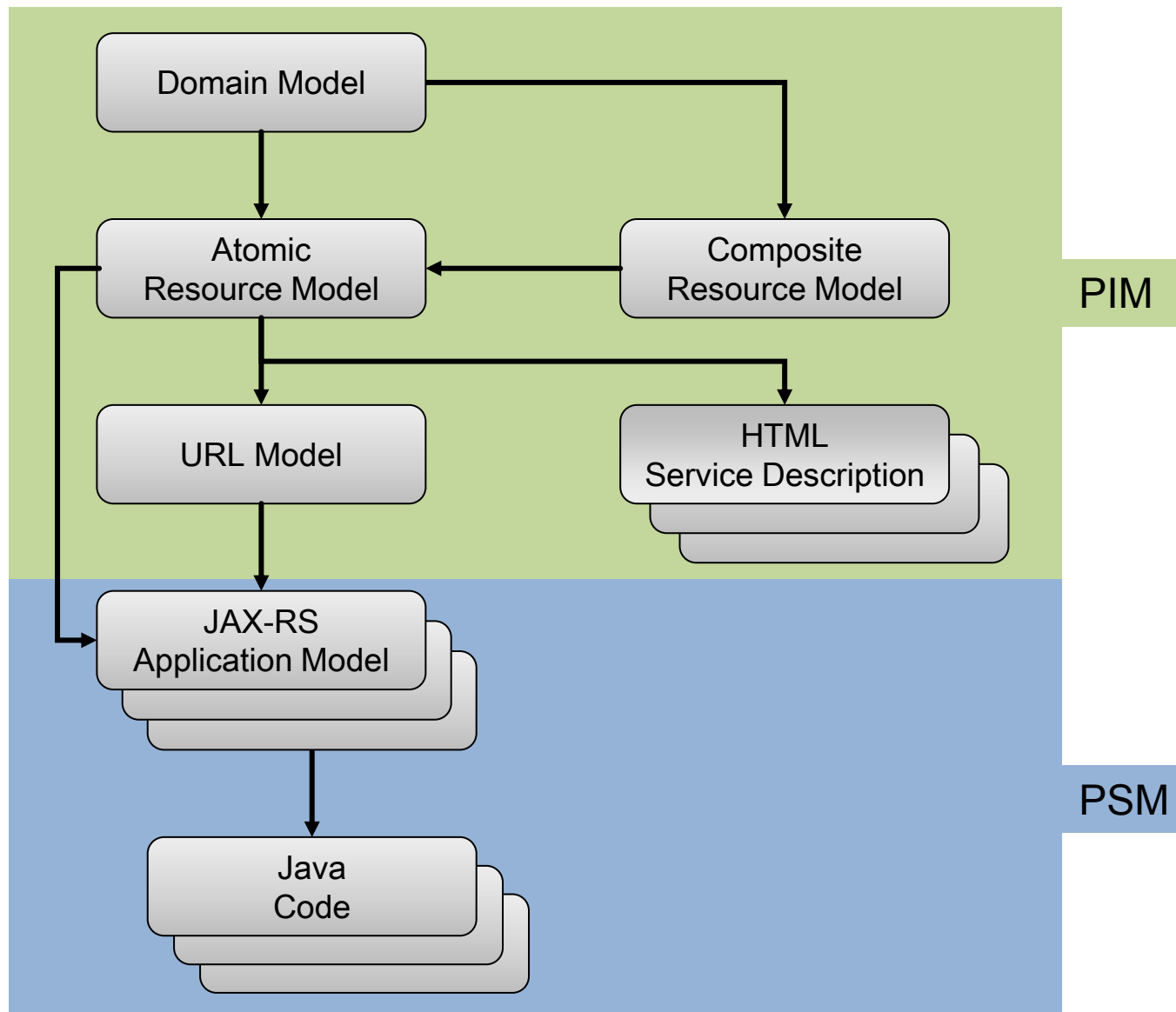
# REST and the WWW – Conclusion

- Some constraints are already fulfilled by the building blocks of the WWW
  - HTTP, URI, MIME
- Some constraints have to be fulfilled by service providers (and consumers)
  - Uniform interface & HATEOAS
- Existing work shows, that constraints that are the responsibility of the service provider are often not fulfilled properly

# Layered Meta-Model [3]

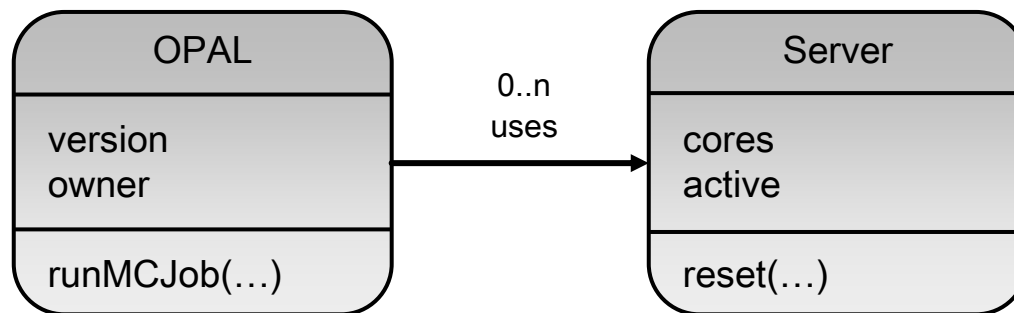
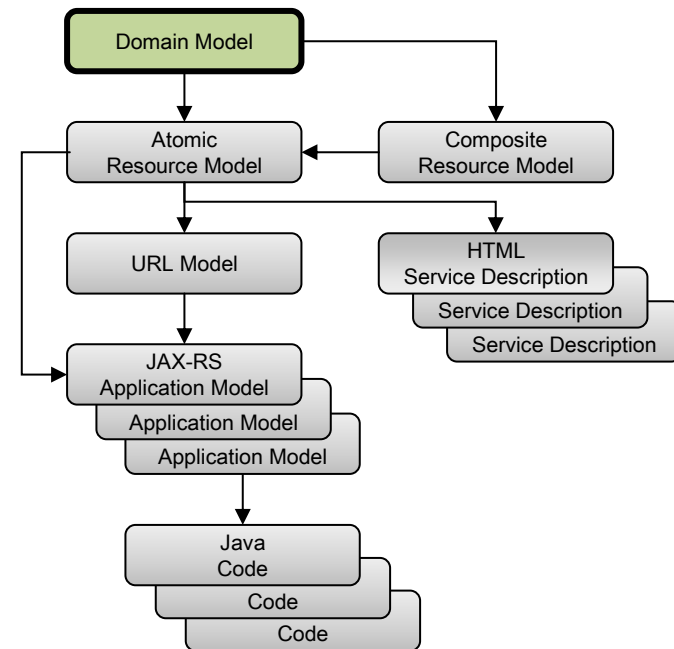


# Meta-model – Overview



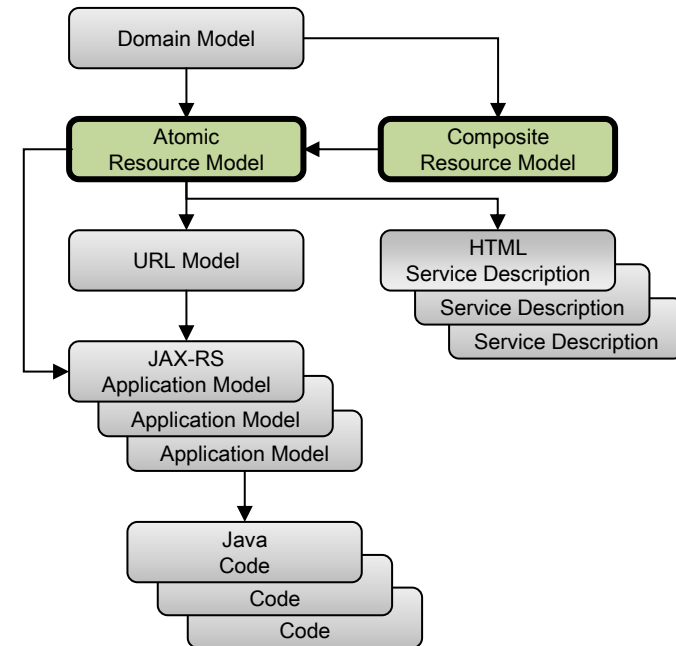
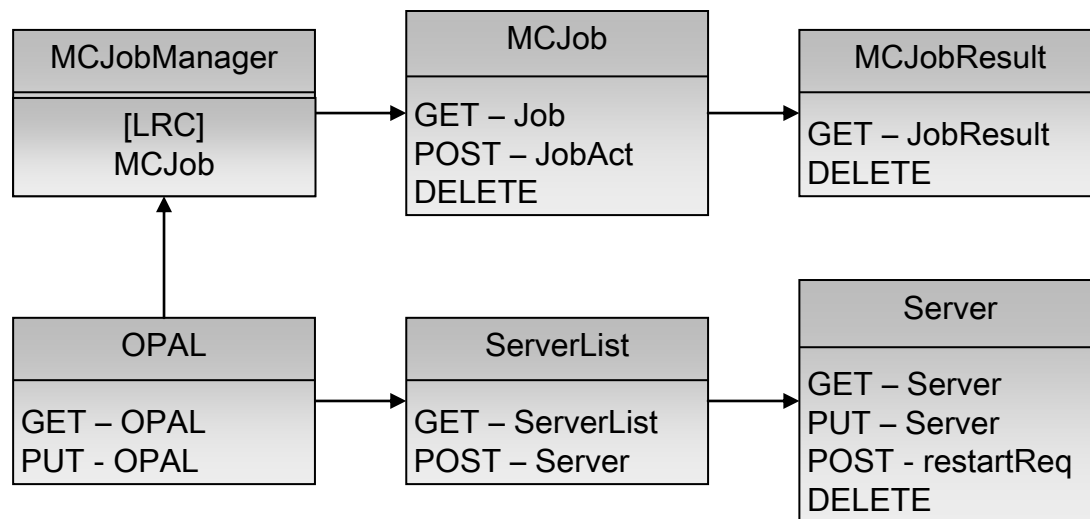
# Meta-model – Domain Model

- Independent of the concept of resources
- Tailored to application domain



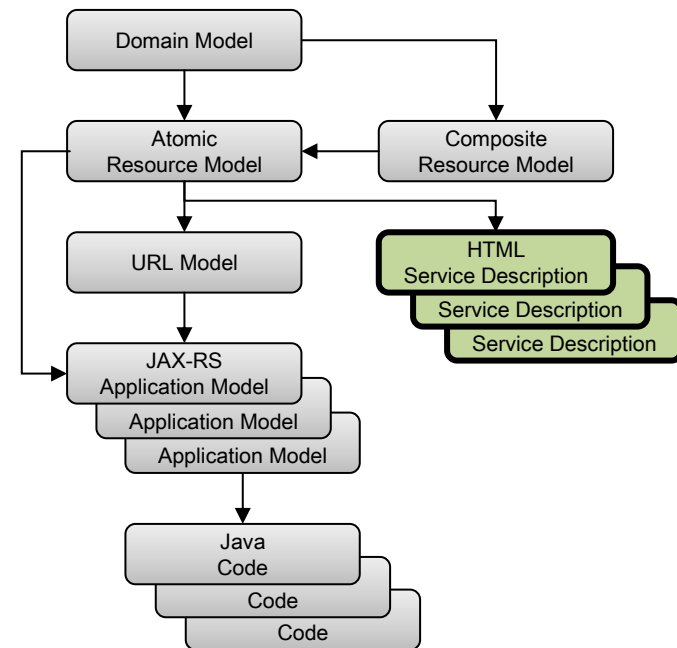
# Meta-model – Resource Models

- Describes the resources provided by a REST service
- Composite Resource Model provides an additional abstraction layer for easier modeling



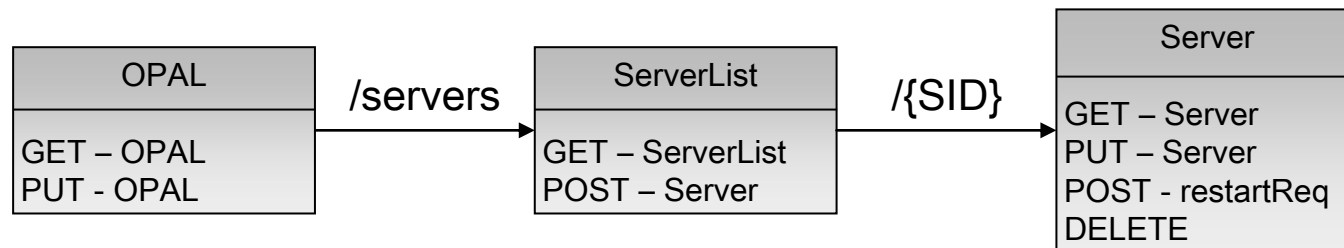
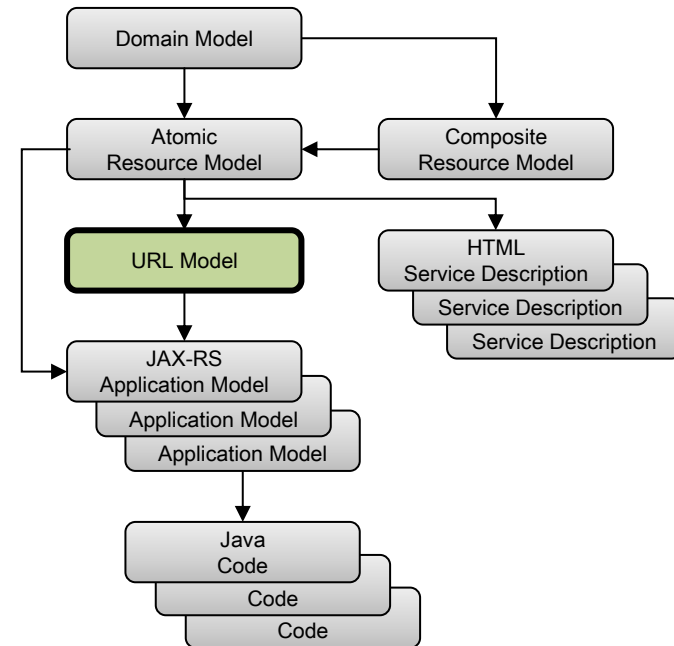
# Meta-model – Service Description

- View on the resource model
- Interface description for service clients
- Independent of URL structure, following HATEOAS



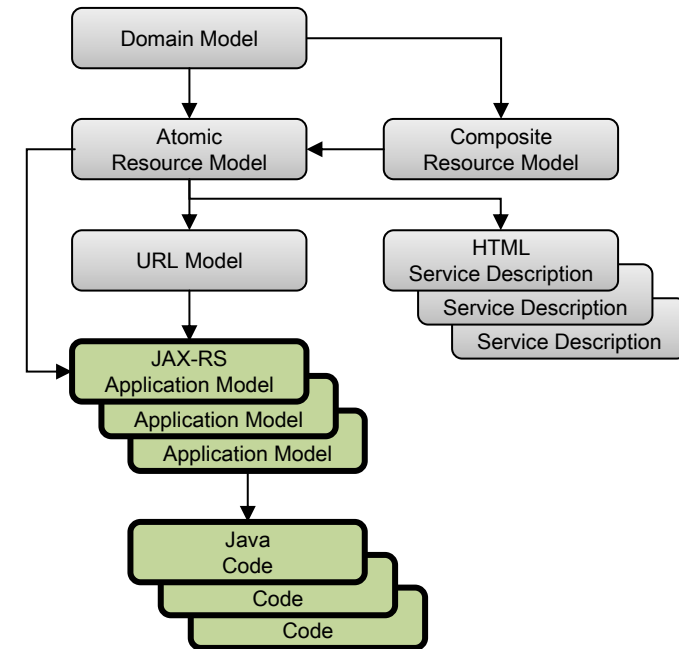
# Meta-model – URL Model

- Assigns URLs to resources
- HATEOAS implies that the URL structure is irrelevant for clients
- Nevertheless, the URL structure is needed for the service implementation



# Meta-model – Application Model and Code

- Application model combines resource model and URL model
- Application model is bound to specific platform
- Generated code has to be extended with domain logic



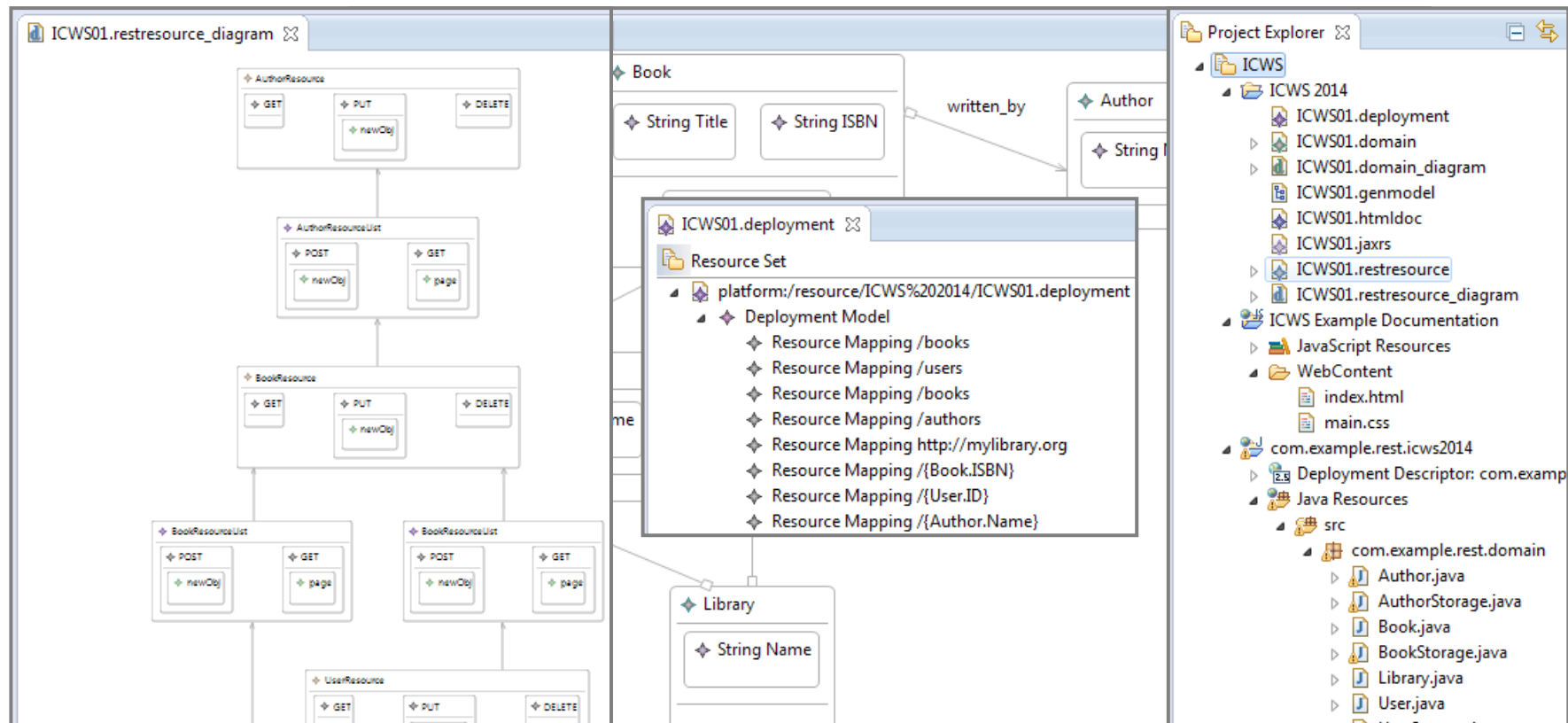
# Prototype



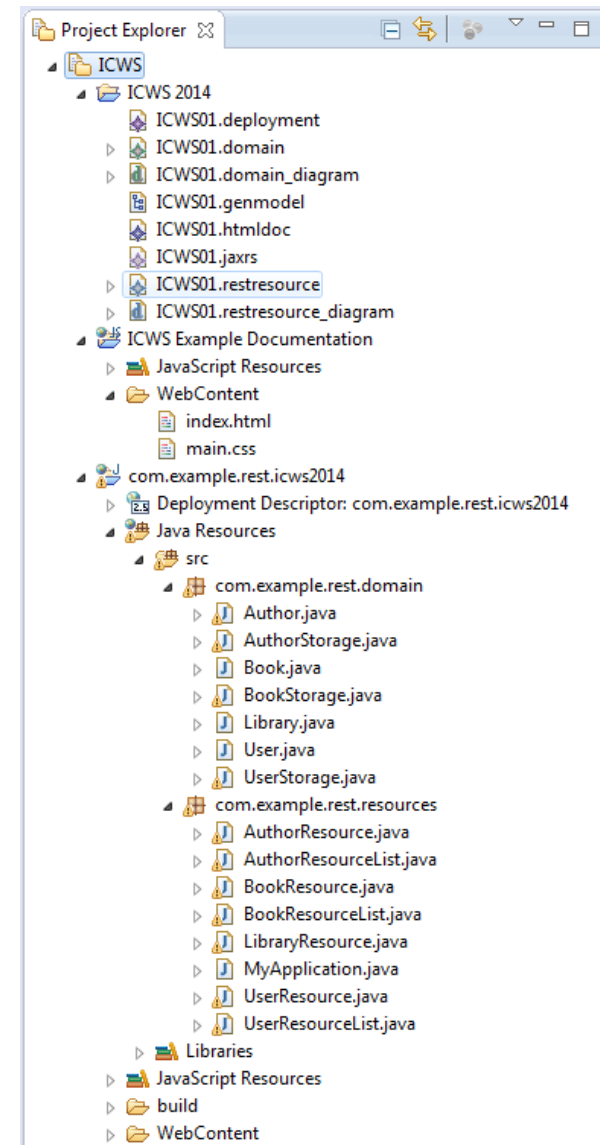
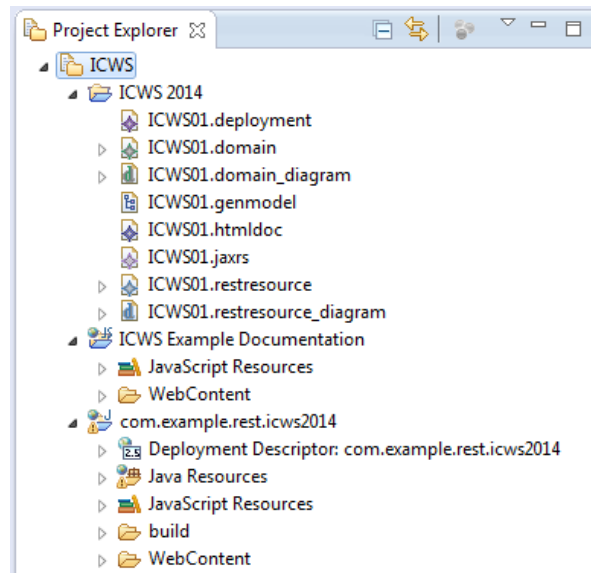
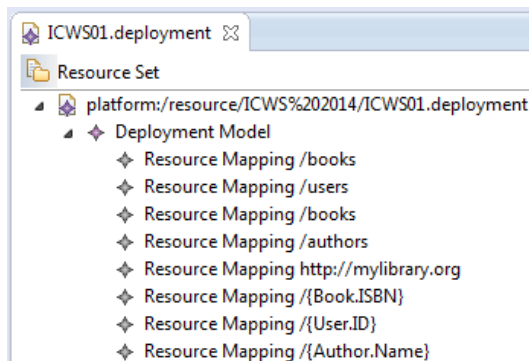
# Prototype – Overview

- Graphical tool supporting model creation, validation and transformation
- Integrated in Eclipse platform based on Eclipse Epsilon (EMF, GMF)
- Supports generation of HTML based documentation
- Supports generation of JAX-RS based implementation

# Prototype – Screenshots



# Prototype – URL Model & Eclipse Integration



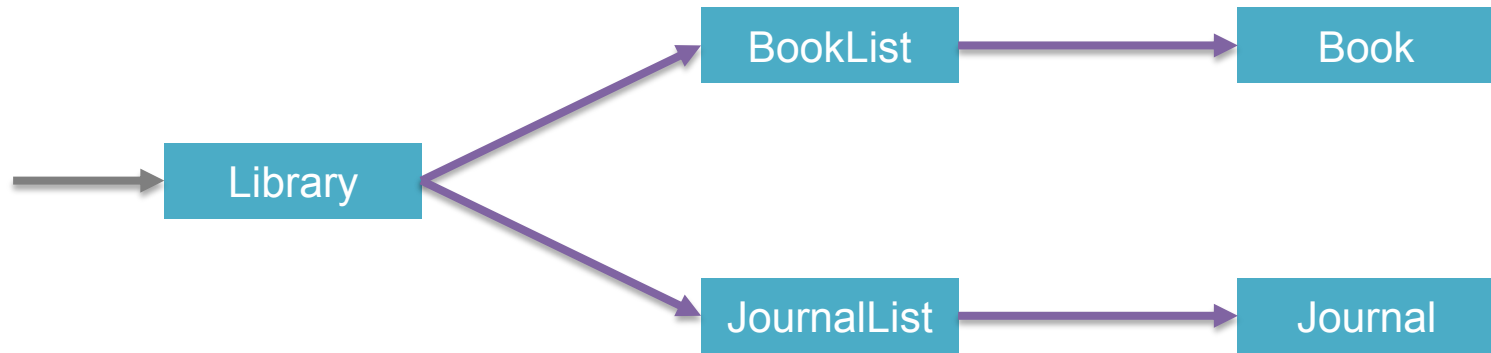
# Formalization



# Resource Graph

- A REST API is a set resources connected with links
  - A graph!
- $G_{\text{RES}} = (V_{\text{RES}}, E_{\text{RES}})$
- $V_{\text{RES}}$  = set of resources (set of  $M_R(t)$  functions)
- $E_{\text{RES}} \subseteq (V_{\text{RES}} \times V_{\text{RES}})$  = set of relations between resources

# Resource Graph – Example



$G_1 = (V_1, E_1)$

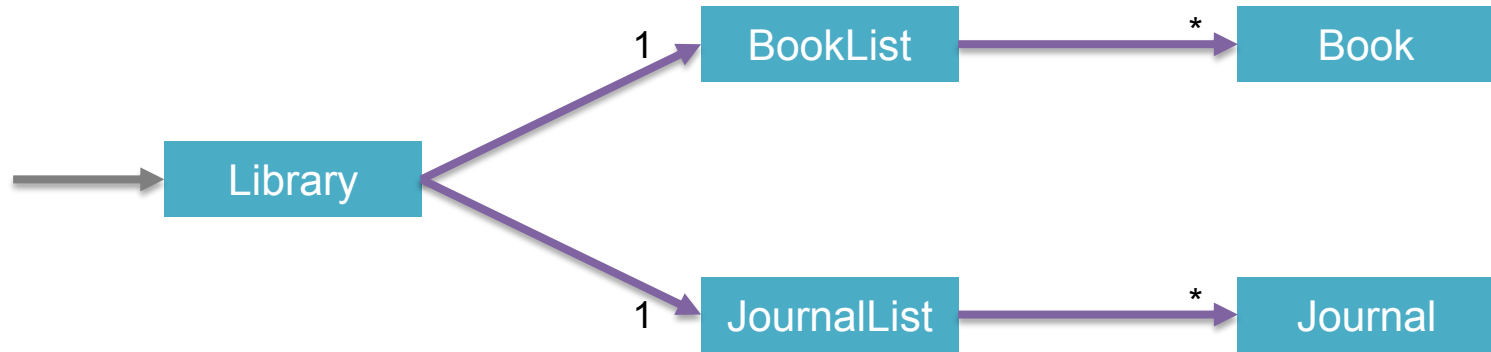
$V_1 = \{\text{Library, BookList, Book, JournalList, Journal}\}$

$E_1 = \{(\text{Library, BookList}), (\text{BookList, Book}), (\text{Library, JournalList}), (\text{JournalList, Journal})\}$

# Relations – Details

- *Card*:  $E_{\text{RES}} \rightarrow \{“1”, “*”\}$ 
  - Assigns a cardinality to each relation
  - Complete mapping, i.e. well defined for each relation
  
- *Type*:  $E_{\text{RES}} \rightarrow P\{“navigation”, “creation”\} \setminus \{\}$ 
  - Assigns one or more types to a relation
    - Set of types is user defined / extensible
  - Complete mapping, i.e. well defined for each relation
    - And mapping to the empty set is not allowed

# Relations – Example



$$G_1 = (V_1, E_1)$$

$$V_1 = \{\text{Library, BookList, Book, JournalList, Journal}\}$$

$$E_1 = \{(\text{Library, BookList}), (\text{BookList, Book}), (\text{Library, JournalList}), (\text{JournalList, Journal})\}$$

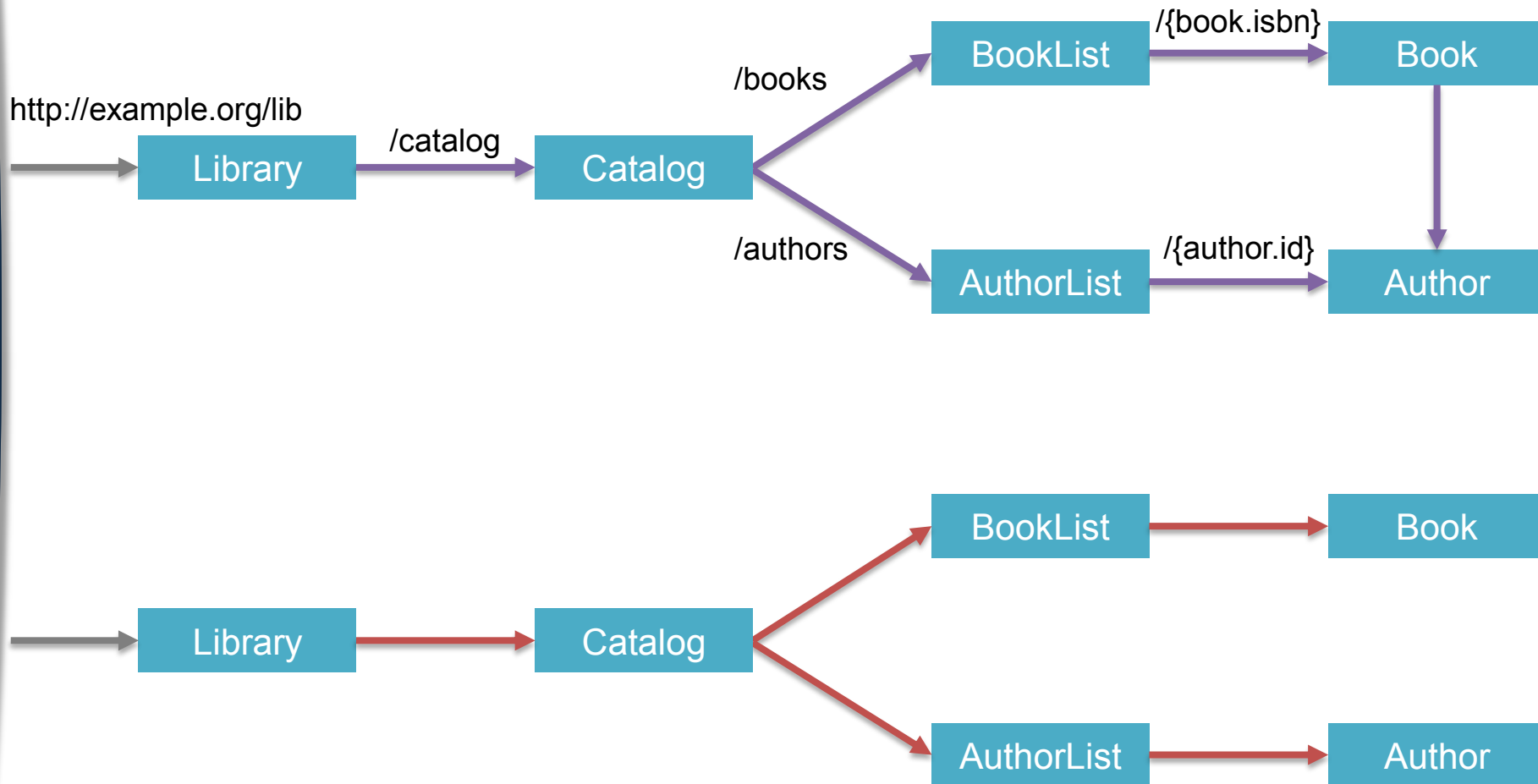
$$\text{Card} = \begin{cases} (\text{Library, BookList}) & \rightarrow "1" \\ (\text{BookList, Book}) & \rightarrow "*" \\ (\text{Library, JournalList}) & \rightarrow "1" \\ (\text{JournalList, Journal}) & \rightarrow "*" \end{cases}$$

$$\text{Type} = \begin{cases} (\text{Library, BookList}) & \rightarrow \{\text{navigation}\} \\ (\text{BookList, Book}) & \rightarrow \{\text{navigation, creation}\} \\ (\text{Library, JournalList}) & \rightarrow \{\text{navigation}\} \\ (\text{JournalList, Journal}) & \rightarrow \{\text{navigation, creation}\} \end{cases}$$

# URI Graph (1)

- $G_{\text{URI}} = (V_{\text{RES}}, E_{\text{URI}})$ 
  - Same nodes like resource graph but different edges
- $E_{\text{URI}} \subseteq E_{\text{RES}}$ 
  - All relations between resources that are annotated with an URI fragment

# URI Graph – Example



## URI Graph (2)

- $Root_{EXPL} = V_{RES} \rightarrow \{\text{true}, \text{false}\}$ 
  - User defined
- $Root_{IMPL} = V_{RES} \rightarrow \{$   
 $\text{true } \forall V \in V_{RES} \mid \forall (S,T) \in E_{URI} : T \neq V,$   
 $\text{false else}\}$ 
  - Resources without any incoming relation with URI annotations
- $V_{RES-ROOT} = \{V \in V_{RES} \mid Root_{EXPL} \vee Root_{IMPL}\}$ 
  - Root resources

## URI Graph (3)

- *URIAnnotation*:

$$E_{\text{RES}} \rightarrow (URI\text{FragmentStatic} \cup URI\text{FragmentDynamic})$$

- URI fragment annotations attached to the relations of the resource graph
- Incomplete, i.e. maybe undefined for a relation
- URIFragmentDynamic = contains variable parts, denoted by {}
- URIFragmentStatic = contains no variable parts

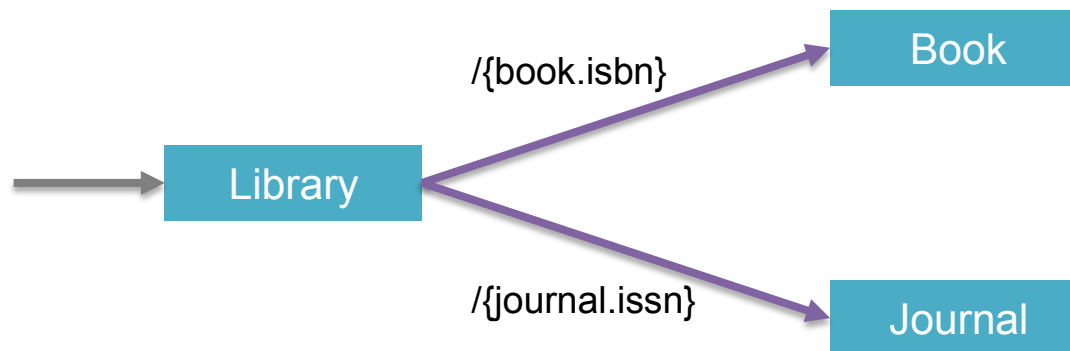
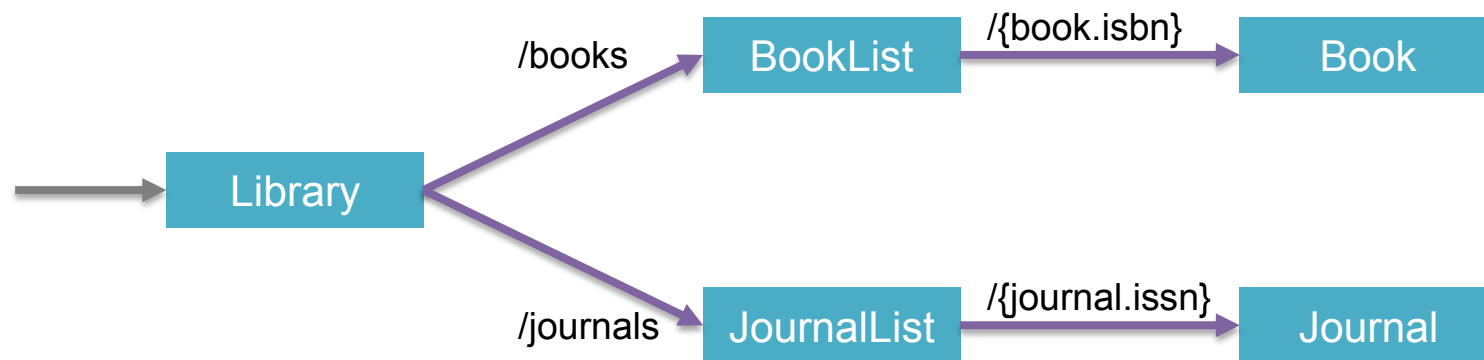
- $E_{\text{URI}} = \{E \in E_{\text{RES}} \mid \text{URIAnnotation}(E) \text{ is defined}\}$

# Use Cases



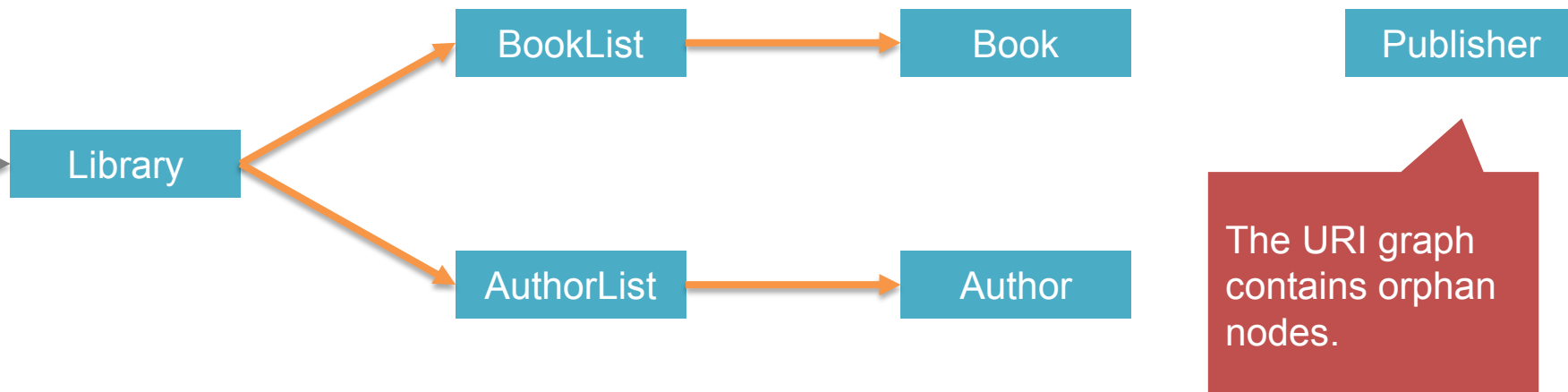
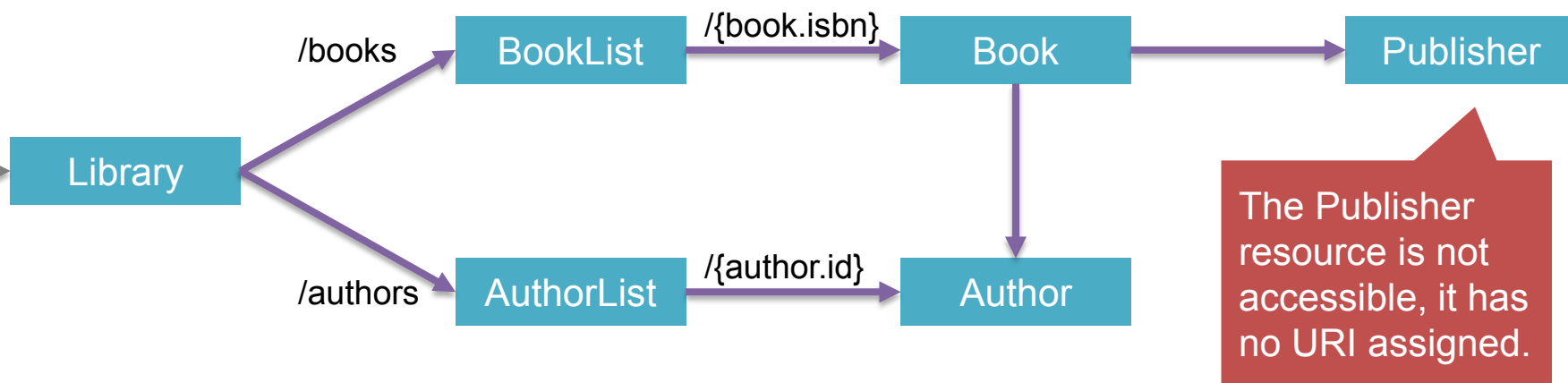
# Detect ambiguity in URL structures

- “Avoid runtime exceptions in JAX-RS”



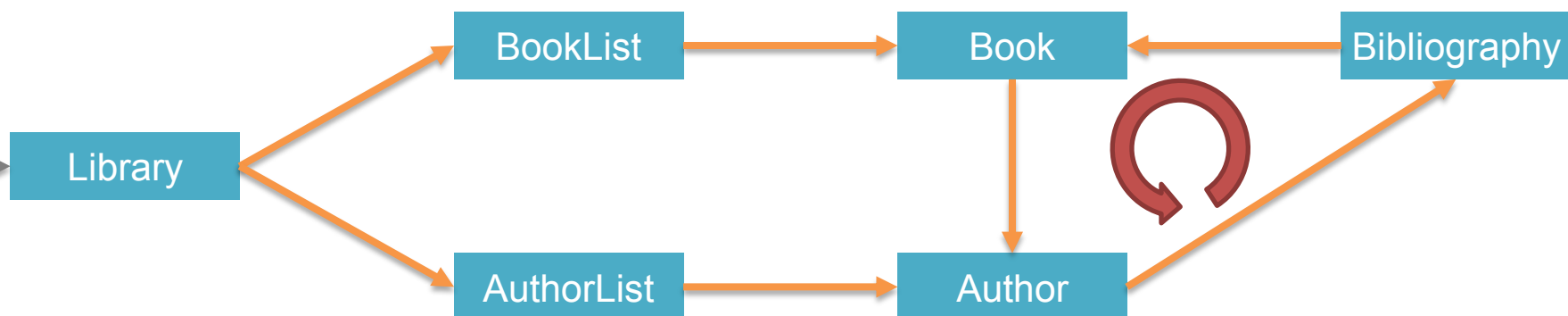
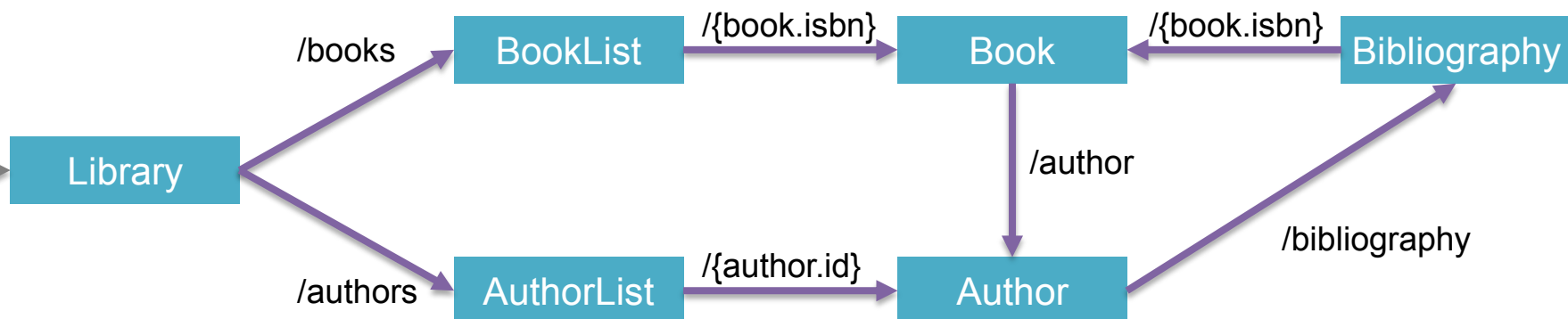
# Check URI mapping for completeness

## ■ “Avoid runtime exceptions in JAX-RS”



# Detect infinite URIs

## ■ “Avoid runtime exceptions in JAX-RS”



<http://example.org/library/books/12/author/bibliography/12/author/bibliography/12/author/...>

# Conclusion



# Conclusion

- Designing REST compliant services is challenging
- We propose to use MDSD techniques to tackle this
- For that we provide
  - A set of meta-models
  - A set of model-to-model transformations
  - A prototype
- Formalization as alternative modeling approach
  - Foundation for model analysis

## Contact

florian.haupt@iaas.uni-stuttgart.de  
www.iaas.uni-stuttgart.de

# References

- [1] Fielding, R.: *REST APIs must be hypertext-driven*. <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>.
- [2] Fielding, R. T.: *Architectural styles and the design of network-based software architectures*. Dissertation, University of California, Irvine, 2000.
- [3] Haupt, F.; Karastoyanova, D.; Leymann, F.; Schroth, B.: *A model driven approach for REST compliant services*. In: Proceedings of the IEEE International Conference on Web Services, ICWS 2014.

