MDD4REST: Model-Driven Methodology for Developing RESTful Web Services

Amirhossein Deljouyi and Raman Ramsin[a]
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Keywords: Model-Driven Development, Domain-Driven Design, Web Engineering, REST Architectural Style, Model Transformation, Automatic Code Generation.

Abstract: Web services based on the REpresentational State Transfer (REST) architectural style have become increasingly popular in recent years. REST provides several desirable features, such as simplicity and scalability; however, developing RESTful web services involves repetitive and trivial tasks that can be avoided through automatic code generation. Model-Driven Development (MDD) can be used to this aim, as it facilitates the construction of complex applications and can provide automatic code generation through transformations of models. This paper presents *MDD4REST* as a model-driven methodology, consisting of a framework and a process, for developing RESTful web services. *MDD4REST* takes advantage of Domain-Driven Design (DDD) to produce a rich domain model for web services. It provides an effective method for designing RESTful web services using modeling languages, and supports automatic code generation through transformation of models. In addition, *MDD4REST* has the capability to support modern web architectures and patterns, such as Microservice, Event-Driven, and CQRS.

1 INTRODUCTION

Offering software in the form of web services has gained immense popularity due to the evolution of cloud architectures. REpresentational State Transfer (REST) comprises a set of rules and practices that provide simple and comprehensible APIs, clear representational structures, and scalable services for use in web services engineering. Due to its simplicity and scalability, the REST architecture has become increasingly popular among web-service developers. Among the architectures used in web service design (REST, WSDL, SOAP), REST is the most common; it has significantly changed how systems are developed based on web services (Fielding and Taylor, 2000; Mulloy, 2013; Ong et al., 2015; Richardson and Ruby, 2008; Rodriguez, 2008).

Model-Driven Development (MDD) is a software engineering approach in which models are construed as primary artifacts of the software development process, from requirements engineering to analysis, design, and implementation. MDD facilitates the construction of complex applications and supports automatic code generation through transformation of models (Hailpern and Tarr, 2006; Siegel, 2014; Truyen, 2006); its potential has therefore been recognized in designing RESTful web services by using modeling languages (Zolotas et al., 2017).

On the other hand, Domain-Driven Design (DDD) is an effective method for producing a rich domain model for web services by focusing on the problem domain. DDD can take advantage of MDD best practices in order to develop a system based on models (Evans and Evans, 2004).

Web services can address functional or non-functional requirements. Functional requirements can be expressed through visual models or text, based on which the domain model is formed. Non-functional requirements can be addressed by designing appropriate architectures. Architectural styles and patterns, including Microservice, Event-Driven, and CQRS, help enhance the scalability and performance of web services (Fowler, 2002, 2017; Greg Young, 2010; Newman, 2015; Rademacher et al., 2017).

Despite their merits, existing MDD methods for web services engineering do not adequately cover the web services development process and fail to produce all of its artifacts. Also, they fail to support high-level modeling at an adequately abstract level, and fail to provide adequate complexity management features.

[a] https://orcid.org/0000-0003-1996-9906

Deljouyi, A. and Ramsin, R.
MDD4REST: Model-Driven Methodology for Developing RESTful Web Services.
DOI: 10.5220/0011006300003119
In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 93-104
ISBN: 978-989-758-550-0; ISSN: 2184-4348

We propose a model-driven methodology, which we have chosen to call MDD4REST, for developing RESTful web services. Modeling levels and model transformation rules are precisely defined in *MDD4REST*, and DDD is applied for producing the domain model. *MDD4REST* has been evaluated by applying four different categories of criteria in order to evaluate its different aspects. Furthermore, it has been empirically validated through application to a web development project in a software development company; this case study has demonstrated the degree of applicability of the proposed methodology, and has helped identify its strengths and weaknesses.

The rest of this paper is structured as follows: Section 2 provides an outline of the related works; Section 3 presents an overview of *MDD4REST*; modeling levels and transformation rules are explained in Section 4; Section 5 presents a process for applying the MDD4REST framework; Section 6 provides the evaluation results; and Section 7 presents the conclusion and a discussion of the future work.

2 RELATED WORKS

In order to elicit the target methodology, the different facets of an MDD methodology for developing RESTful web services have been studied. In this section, existing approaches are briefly reviewed, along with three high-level frameworks for MDD and web engineering.

2.1 Existing MDD/DDD Approaches for Developing RESTful Web Services

Valverde and Pastor have produced a methodology by extending the OO-Method methodology with a basic meta-model for generating RESTful web services (Valverde and Pastor, 2009). Schreier has introduced meta-models for modeling structural and behavioral aspects of RESTful applications (Schreier, 2011). Haupt et al. have presented a meta-model for REST constraints (Haupt et al., 2014). Ed-Douibi et al. have proposed a method for developing RESTful web services by taking advantage of MDD and combining REST principles with the Eclipse Modeling Framework (EMF) (Ed-Douibi et al., 2015); EMF allows the construction of meta-models using the Ecore language (Steinberg et al., 2008). Ed-Douibi et al. have also addressed various challenges of RESTful web services development, including testing and integration of APIs, through MDD (Ed-Douibi, 2019).

Zolotas et al. have proposed a methodology for generating RESTful web services based on software requirements (Zolotas et al., 2017); in this approach, informal specifications or use-cases are used for modeling functional requirements, and behavioral requirements are modeled in activity diagrams or storyboards. Gonçalves and Azevedo have introduced a model-driven approach in which a DSL is proposed for developing OpenAPI specifications; this enables developers to utilize the first-design approach, focusing on the definition of resources and relationships (Gonçalves and Azevedo, 2018). Koren and Klamma have developed a method for creating front-end pages from OpenAPI specifications (Koren and Klamma, 2018). Hernandez-Mendez et al. have proposed an MDD method for consumption of RESTful APIs in single-page applications, which aggregates RESTful APIs through a query service meta-model (Hernandez-Mendez et al., 2018).

Jegadeesan has proposed an approach for generating web services of various granularities in which requirements are obtained through DDD (Jegadeesan, 2009). Kapferer has introduced a DDD-driven approach for strategic design of systems and decomposition of web services (Kapferer, 2020). Terzić et al. have addressed the challenges of RESTful web services in a microservice architecture (such as routing, and microservice auto-discovery and registering) (Terzić et al., 2018).

2.2 High-level Process Frameworks for MDD and Web Engineering

We have used three high-level process frameworks as bases for developing our proposed methodology; these frameworks are briefly introduced in this section.

Babanezhad et al. have proposed a high-level process framework consisting of process patterns for developing web-based systems (Babanezhad et al., 2010). By applying abstraction to existing MDD methodologies, Asadi et al. have devised a generic framework of process patterns for MDD (Asadi et al., 2010). Blake has provided a lightweight framework for development of web services (Blake, 2006).

3 OVERVIEW OF PROPOSED MDD4REST METHODOLOGY

As mentioned before, *MDD4REST* consists of two parts, a modeling framework (shown in Figure 1) and a process.

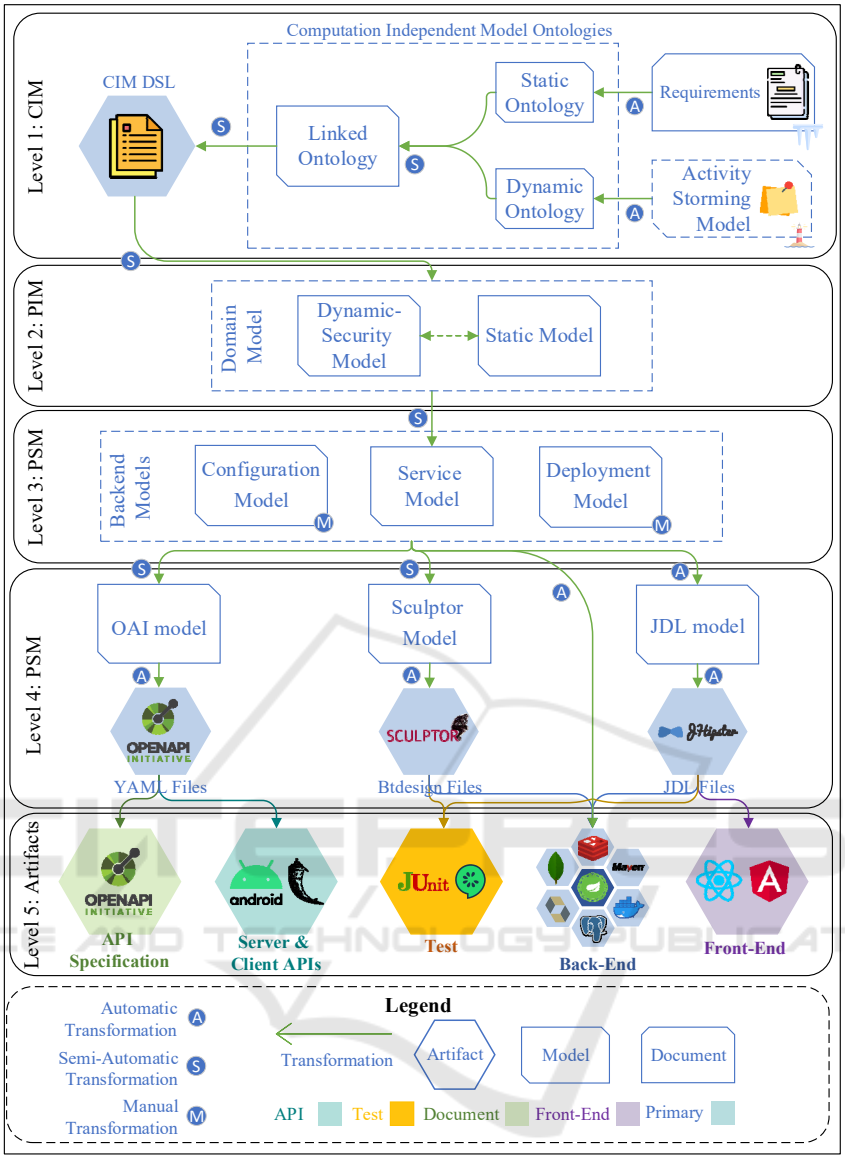

Figure 1: MDD4REST modeling framework.

The modeling framework supports modeling the target web system. It consists of four levels of abstraction and was defined based on the results of the literature review. A fifth level houses the artifacts of the end product (components of the generated system). The framework provides a set of model transformations that supports semi-automatic construction of target models from source models, producing the detailed design of RESTful web services. All modeling levels are thoroughly described in Section 4. A process was also proposed for applying the *MDD4REST* framework that will be explained in Section 5.

Software Architectures. The *MDD4REST* framework supports various architectural styles and patterns for complex web applications. The system generated through applying the *MDD4REST* framework will be based on an onion architecture. Code generators support architectural styles such as Microservice and Event-Driven. Also, command and query concepts are split from the first level to support more advanced patterns, including CQRS.

Tool Support. Several tools have been developed to facilitate using the *MDD4REST* framework; these tools are depicted in Figure 2: 1) *mdd4rest-annotater*, which is mainly based on BRAT (Stenetorp et al.,

2012), annotates software requirements and generates the static ontology; 2) *mdd4rest-activity-storming*, which is based on Eugenia (Kolovos , 2017), provides a modeling tool and meta-model for exploring, visualizing, designing, and formalizing the business domain; 3) *mdd4rest-generator*, which is mainly based on the Eclipse Epsilon family of model transformation and management languages (including ETL, EOL, EML, and EGL), provides a set of transformation rules for generating models; 4) *mdd4rest-cli* is a command-line interface for facilitating the use of the *mdd4rest-generator*; 5) *mdd4rest-metamodels* contains a set of EMF meta-models for the framework's modeling levels. All the projects for *MDD4REST*, and the case study's artifacts, are available online (Deljouyi, 2021).

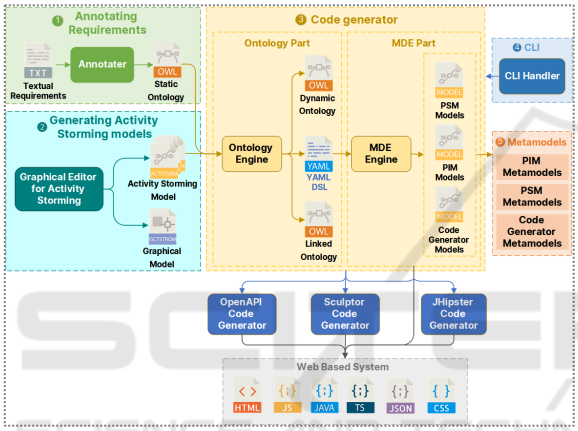

Figure 2: MDD4REST Framework Tools.

4 MDD4REST FRAMEWORK

This section introduces the four modeling levels of the *MDD4REST* framework. Examples of the models produced are presented in Section 6.1 (Case Study).

4.1 Level 1: Computation-Independent Model (CIM)

At level 1, system requirements are identified and modeled, and the system's domain model is thus formed. Two approaches are used at the first level for domain modeling: textual requirements specifications and activity storming models.

4.1.1 Textual Requirements Specifications

Due to the fact that textual specifications of the requirements are prevalently utilized in software development projects, they are good sources for structural modeling of a system. *mdd4rest-annotator* can annotate textual requirements and parse annotations into a static ontology (Figure 3).

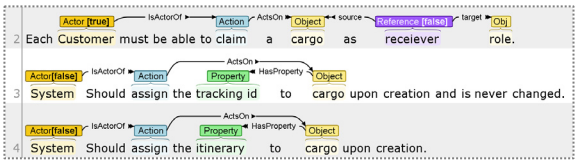

Figure 3: An example of textual requirements specifications and annotations.

4.1.2 Activity Storming Diagrams

For behavioral modeling, we have developed a new diagram named "Activity Storming" via combining UML Activity Diagram elements with concepts taken from Event Storming (Brandolini, 2013). Activity diagrams provide formalization, and Event Storming covers DDD concepts. Activity Storming thus provides both formalization and support of DDD concepts. In addition, our *mdd4rest-activity-storming* tool comes as an Eclipse plugin to explore, visualize, and design Activity Storming models. The elements of Activity Storming are shown in Figure 4.

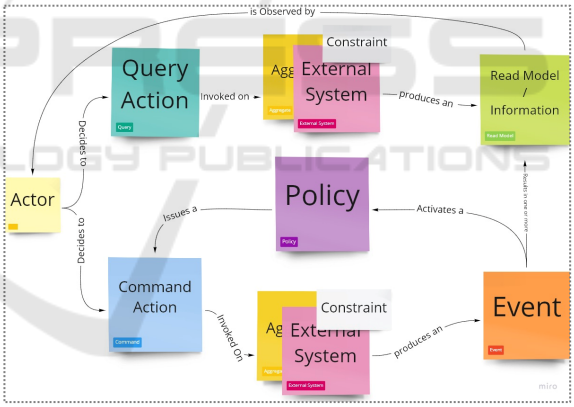

Figure 4: Elements of Activity Storming.

4.1.3 Ontologies

MDD4REST incorporates three ontologies regarding different views of the system: 1) *Static Ontology,* representing a structural view; 2) *Dynamic Ontology,* representing a dynamic view; 3) *Linked Ontology,* combining structural and dynamic views. *mdd4rest-generator* transforms activity storming models into a dynamic ontology, and then aggregates the static and dynamic ontologies to produce a linked ontology. Ultimately, *mdd4rest-generator* transforms the linked ontology into a YAML-based Domain-Specific Language (DSL).

4.2 Level 2: Platform-Independent Model (PIM)

The focus of level 2 is on the REST architecture and contains resource-oriented concepts. Two models are produced at this level: Static and Dynamic-Security models. At this level, *mdd4rest-generator* semi-automatically transforms the YAML-based DSL produced at level 1 into the models of level 2. Figure 5 shows the architecture of this level. As Static and Dynamic models have common elements, EOL rules are developed for synchronizing them.

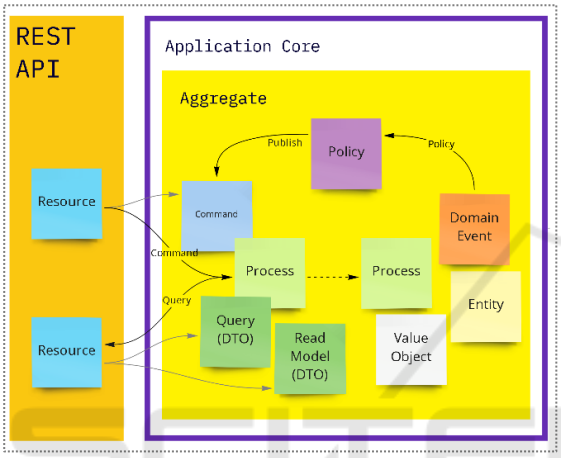

Figure 5: PIM's architecture.

4.3 Level 3: Platform-Specific Model (PSM) of the Architecture

Levels 3 and 4 are both platform-specific. At level 3, the architecture of the system will be determined; PIM models are merged into a service model, and the user can create deployment and configuration models manually. There are two types of architectural views at the third level: *intra-application* and *inter-application*. *Intra-application* represents the architecture of each application, which may be simple or CQRS. *Inter-application* describes the architecture between the applications, as several microservice applications can comprise the system. Figure 6 and Figure 7 demonstrate intra-application and inter-application views, respectively.

4.4 Level 4: Platform-Specific Model (PSM) for Code Generators

Level 4 contains models and DSLs for code generators, namely Jhipster, Sculptor, and OpenAPI. *mdd4rest-generator* transforms level-3 models into code generator models, and DSLs are then generated by applying EGL rules on these models. Code generators use these level-4 products to generate the final artifacts of the target system (residing at level 5).

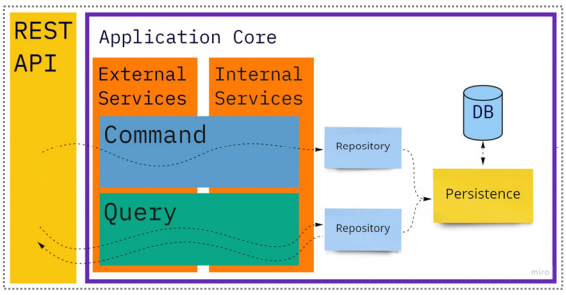

Figure 6: CQRS Intra-Application view.

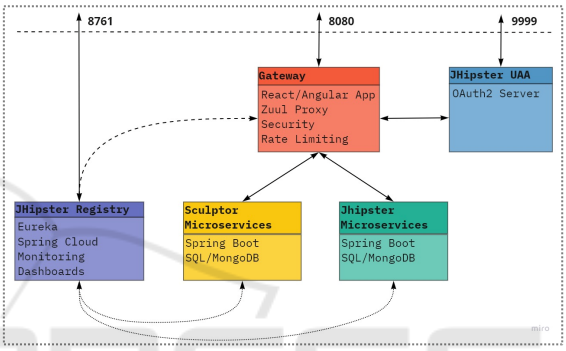

Figure 7: Inter-Application architecture view.

5 MDD4REST PROCESS

In order to define an MDD methodology, a process must be defined for applying the modeling framework. The process that we propose (shown in Figure 8) consists of four phases: *Start-up*, *Construction*, *Transition*, and *Maintenance*. The process is primarily based on the Web Engineering Process Framework of (Babanezhad et al., 2010). The phases are performed serially, but they are broken down into stages that are executed iteratively.

The goal of the Start-up phase is to acquire knowledge about the target system and perform the essential activities for initiating the project.

The Construction phase is aimed at developing the target system in several iterations, and consists of three coarse-grained stages: *Analysis*, *Model-Driven Development*, and *Implementation*. Construction is where modeling is performed based on the different levels of the *MDD4REST* framework. In the Analysis stage (level 1), the functional and non-functional requirements of the system are elicited, estimation and prioritization are performed, and higher-priority requirements are selected for the iterations.

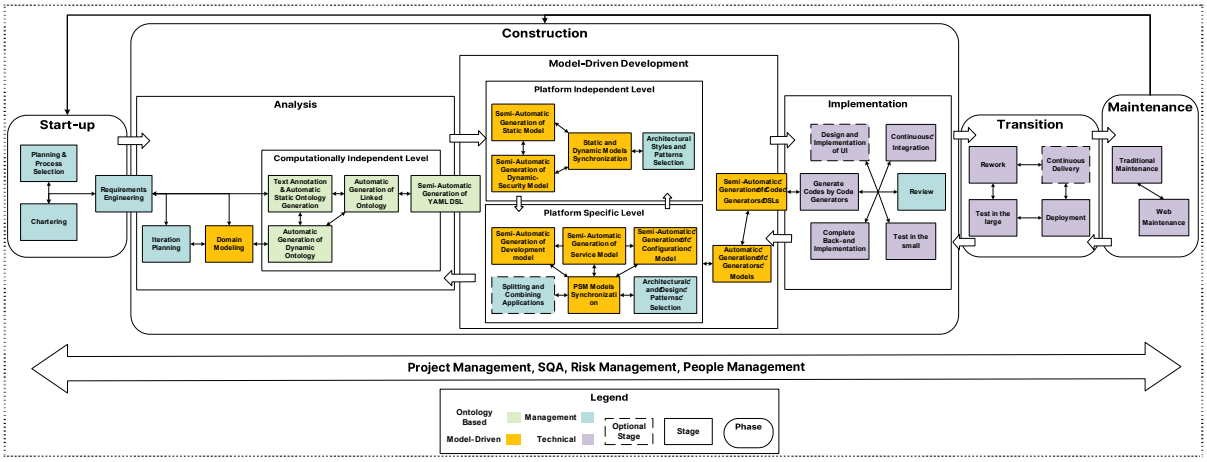

Figure 8: MDD4REST process.

In the Model-Driven Development stage, activities related to levels 2 to 4, including the transformations, are performed. In the Implementation stage, code is generated by code generators, and is then completed by developers (level 5).

The Transition phase is focused on deploying the developed system into the user environment. The Maintenance phase focuses on maintenance and support activities. Umbrella activities are also considered in this process, shown on the arrow at the bottom of Figure 8.

Two types of roles are involved in this process: *mandatory* and *optional*. The mandatory roles are: *user, product owner, coach, domain expert,* and *developer*. The optional roles are: *architecture owner, tester, modeler, user interface designer,* and *infrastructure expert*.

6 EVALUATION

To assess the applicability and effectiveness of *MDD4REST*, a case study was conducted and criteria-based evaluation was applied. The research questions were as follows: **RQ1.** How logical and accurate are the modeling levels and model transformations? **RQ2.** Does *MDD4REST* facilitate the design and development of RESTful web services? **RQ3.** Are the concerns of RESTful web service well covered? **RQ4.** How applicable are the tools in *MDD4REST*?

6.1 Case Study

The case was a web development project in a software development company that specializes in producing software solutions for medium to large businesses. *MDD4REST* was used to develop a virtual gift card generator system. A technical manager of the company was actively involved in the study, both as product owner and user. The first author was involved in the project and carried out the activities prescribed by *MDD4REST* in collaboration with the technical manager. The development took about 30 working days, and all the products were delivered to the technical manager to acquire his feedback and confirmation. A questionnaire was designed to collect the final feedback from the technical manager.

6.1.1 Definition

The virtual gift card generator system provides the following services: 1) browsing gift card designs and categories; 2) buying gift cards; 3) receiving the list of orders of a customer; 4) setting a second password for a gift card.

6.1.2 Requirements Engineering

Requirements were elicited and expressed as textual specifications and event storming models. An excerpt of the event storming model of the "buying a gift card" scenario is depicted in Figure 9.

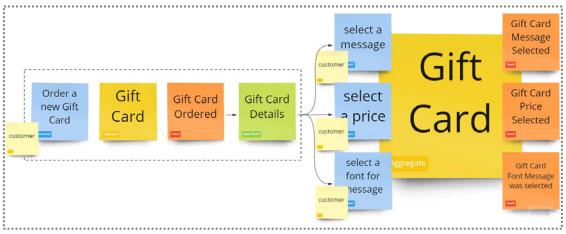

Figure 9: Scenario for buying a gift card.

6.1.3 Level 1

Text Annotation and Automatic Generation of Static Ontology. The textual specifications of requirements were annotated using *mdd4rest-annotator*; an excerpt of the annotation result is shown in Figure 10. The static ontology (Figure 11) was later generated by *mdd4rest-annotator*.

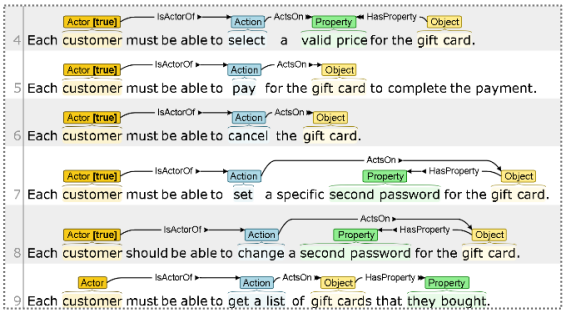

Figure 10: An excerpt of the annotation of the system.

Domain Modeling and Automatic Generation of Dynamic Ontology. Problem domain modeling was performed by designing Activity Storming models. Event Storming models can be used for producing Activity Storming models; however, Event Storming models are optional, and Activity Storming models can be designed directly. The dynamic ontology (Figure 11) was later generated.

Automatic Generation of Linked Ontology. After creating the static and dynamic ontologies, *mdd4rest-generator* merged them into the linked ontology. Figure 11 shows the mapping of static and dynamic ontologies onto the linked ontology. The "property" attribute is mapped from the static ontology to the linked ontology in parts 1 and 3, and the "event" element is mapped from the dynamic ontology to the linked ontology in parts 2 and 4.

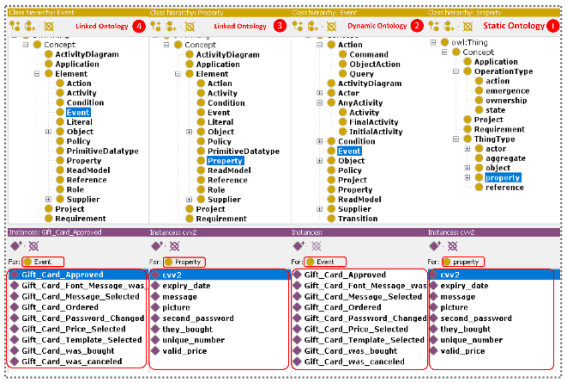

Figure 11: Static, Dynamic, and Linked ontologies.

Semi-automatic Generation of YAML DSL. The mdd4-rest-generator transformed the linked ontology into a YAML-based DSL. This DSL must be reviewed and evaluated and is considered as the system's high-level design.

6.1.4 Level 2

The YAML DSL was given as input to the model-driven engine of *mdd4rest-generator*, and the Static and Dynamic-Security models were generated. The attributes and relationships of the domain and aggregation objects are modeled in the Static model, whereas processes and operations are modeled in the Dynamic-Security model. In this phase, the generated resources are completed; the relationships between resources, operations, and processes are defined. In addition, policies for accessing the resources and the access level of roles are modeled in the Dynamic-Security model. An excerpt of the generated Static and Dynamic-Security models is shown in Figure 12.

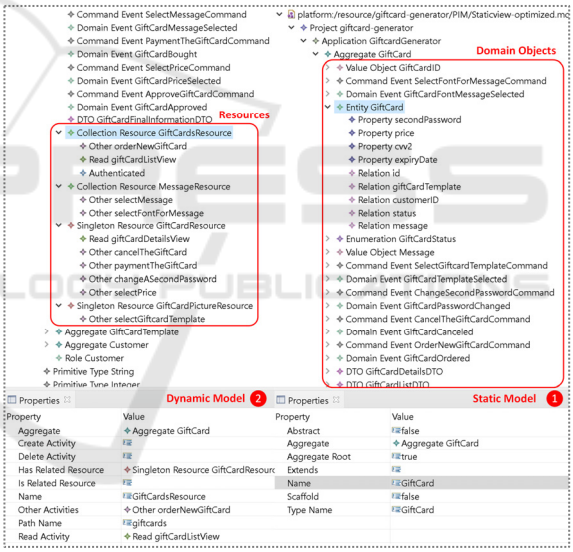

Figure 12: Static and Dynamic-Security models.

6.1.5 Level 3

At this level, *mdd4rest-generator* merged the Static and Dynamic-Security models and transformed them into PSM models (Service, Configuration, and Deployment). In order to generate a system with the CQRS architecture, three modules, namely command, query, and web, were generated as constituents of the Service model (partially shown in Figure 13). Consequently, the domain objects, operations, internal/external services, and resources were completed; the Configuration and Deployment models were enhanced accordingly.

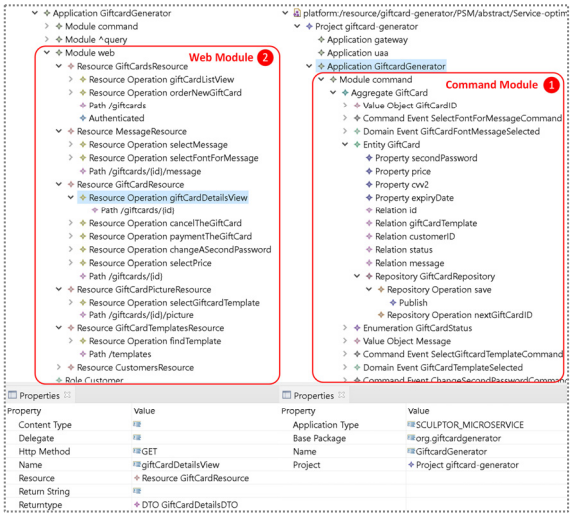

Figure 13: Modules in the Service model.

6.1.6 Level 4

At this level, level-3 models were transformed into models specific to each supported code generator, and DSLs for each code generator were subsequently generated. These DSLs are reviewed and modified as necessary. An excerpt of the Sculptor's DSL is shown in Figure 14.

```
Entity GiftCard {
    String secondPassword nullable
    Double price nullable
    String cvv2 nullable
    Integer expiryDate nullable
    Long giftCardTemplateID nullable
    - @GiftCardID giftCardId key
    - @GiftCardStatus status
    - @Customer customerID nullable
    - @Message message nullable

    def Boolean updateStatus();
    def Boolean isPending();
    def Boolean isBought();
    def Boolean isCanceled();
    def Boolean isApproved();
    def void setStatusPending();
    def void setStatusBought();
    def void setStatusCanceled();
    def void setStatusApproved();

    Repository GiftCardRepository {
        gap
        findByKey;
        @GiftCardSavedEvent  save(@GiftCard  entity)
        publish to GiftCardStoreChannel gap;
        @GiftCardID nextGiftCardID();
                String generateCVV2();
                Integer generateExpiryDate();
    }
}
```

Figure 14: The DSL of the Sculptor Code generator.

6.1.7 Implementation

At this stage, the code was generated by the three code generators, and the body of functions of the generated back-end code was completed. In order to access the front-end services, a single-page application was generated by the Jhipster code generator, which was then improved. Figure 15 shows the appearance of the gift-card generator system.

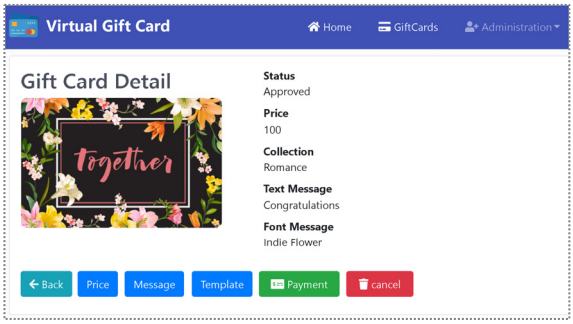

Figure 15: Appearance of the single-page application for the gift-card generator system.

6.1.8 Questionnaire-based Interviews

The participants involved in the case study were interviewed using a specially designed questionnaire. The questionnaire was designed to summarize the opinions of the participants, particularly regarding the research questions; a partial view of the questionnaire is shown in Table 1. Furthermore, the advantages and drawbacks of *MDD4REST* were discussed with the participants for future improvement.

6.2 Evaluation Criteria

We developed a set of evaluation criteria to assess *MDD4REST* in a systematic manner, particularly regarding the research questions. There are four groups of criteria: *general criteria, model-driven criteria, model-driven web engineering criteria, and RESTful web services criteria*. As some groups are quite populous (containing more than 20 criteria), showing the whole sets of results is not possible in this paper. However, excerpts of the criteria and evaluation results are shown in Tables 2 to 5.

6.3 Analysis of Results

The analysis results are presented in the order of the research questions:

Answering RQ1- Adequacy of *MDD4REST* from an MDD perspective: the evaluation results shown in Tables 3 and 5 indicate that modeling levels are defined accurately and distinguishably. However, support for round-trip engineering is low.

Answering RQ2- Usability of *MDD4REST*: the case study demonstrated that *MDD4REST* improves

quality and comprehensibility in design and modeling. As a result of these features and the automation provided, the web services development process is significantly easier to use in comparison with common approaches; this was confirmed by the results of the questionnaire-based interviews.

Answering RQ3- Addressing the concerns of RESTful web services development: the results obtained from criteria-based evaluation (Table 4) and questionnaire-based interviews indicate that there is an adequate level of coverage. The primary deficiency is the maintenance of the generated systems. This issue will be tackled by supporting round-trip engineering in future work.

Answering RQ4- Applicability of *MDD4REST* tools: the results obtained from criteria-based evaluation (Tables 3 and 4) and questionnaire-based interviews indicate that the tool support is adequate, and provides convenient employment of *MDD4REST* at all levels. In fact, tool support is one of the strengths of *MDD4REST*.

6.4 Comparative Analysis

MDD4REST has overcome many of the shortcomings of previous methods. Here are a few issues that have been addressed:

Coverage of web services development lifecycle- Previous methods for generating RESTful web services either do not provide full coverage of the web services development lifecycle, or fail to define a development process at all. In contrast, *MDD4REST* incorporates a detailed process that covers the entire lifecycle, from analysis to maintenance.

Coverage of MDD abstraction levels- Existing methods lack sufficient modeling and abstraction levels, and only a few of them support the CIM Level. In contrast, the *MDD4REST* modeling framework supports modeling at all levels, including CIM, PIM, and PSM.

Rich domain model- Existing methods cover CRUD operations only, whereas *MDD4REST* is not limited to CRUD operations. A rich domain model is produced in *MDD4REST* based on DDD concepts.

Generating the artifacts necessary for RESTful web services- Existing methods do not create all the products needed for web services development based on the REST architecture. In contrast, *MDD4REST* generates code, API specifications, and tests.

Supporting common architectures- Existing approaches do not address the architectures common in modern web systems, such as microservices and event-driven. In contrast, *MDD4REST* covers prevalent architectural styles and patterns.

7 CONCLUSIONS

The *MDD4REST* methodology was developed as a comprehensive model-driven methodology for developing RESTful web services. It is comprehensive in that it provides a multilevel modeling framework along with a process for applying it. Transformation rules have been implemented to generate the models so that the transitions between modeling levels are smooth and trouble-free. Furthermore, we have developed several tools to support *MDD4REST*. For future work, we plan to support other diagrams for domain modeling, improve the textual annotation process by using NLP methods, cover strategic concepts of DDD, and support reverse-engineering from DSLs to models.

Table 1: Part of the designed questionnaire.

General evaluation of the approach		Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1	The approach can be used easily in organizations.					
2	Others can extend the approach.					
3	The approach can improve the quality of code and decrease errors in web services.					
4	Technical people are enthusiastic about this approach.					
Technical evaluation of the approach		**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
1	The approach helps developers and designers significantly.					
2	The approach designs web services adequately.					
3	The approach deploys the systems efficiently and automatically.					
4	The approach can maintain the systems efficiently.					
5	The developed tools are able to improve the automation of the transformation and generation of models.					
6	Common architectures in web engineering are supported.					

Table 2: Partial view of the results of assessment based on General methodology evaluation criteria.

Criterion		Result	Description of possible values
Lifecycle Coverage	Requirements Engineering	**A**	**A**: Supported with detailed instructions. **B:** Supported with general guidelines. **C:** Not Supported.
	Analysis	**A**	
	Design	**A**	
	Implementation	**A**	
	Test	**B**	
	Deployment	**B**	
	Maintenance	**B**	
Coverage of Umbrella Activities	Project Management	**B**	**A**: Supported with detailed instructions. **B:** Supported with general guidelines. **C:** Not Supported.
	Risk Management	**B**	
	Quality Management	**A**	
Clarity of Development Process Definition		**A**	Work-products, actors, and activities are: **A:** entirely supported and precisely described. **B:** partially supported or just mentioned. **C:** weakly supported.
Seamlessness and Smoothness of Transition between Phases		**A**	**A:** Both provided; **B:** Only seamlessness provided. **C:** Only smoothness provided. **D:** None provided.
Encouragement of Active User Involvement		**A**	**A:** The methodology explicitly provides an atmosphere in which the user is actively involved. **B:** The methodology provides a number of guidelines for involving the user. **C:** The methodology does not provide any support.
Manageability of complexity		**A**	**A:** The methodology explicitly provides coping mechanisms. **B:** Some guidelines are defined in order to manage complexity. **C:** This feature is weakly supported.
Process Definition Type		**Process-Centered**	**Process-Centered:** The lifecycle phases, stages, and activities are considered, and other aspects are described as secondary. **Product-Centered:** Products are taken into consideration, and other aspects are described as secondary. **Role-Centered:** roles are taken into consideration, and other aspects are described as secondary.

Table 3: Partial view of the results of assessing support for Model-Driven Development.

Criterion	Result	Description of possible values
Transparency of modeling levels	**A**	The boundary between levels: **A:** is accurately distinguishable. **B:** is relatively transparent. **C:** cannot be distinguished.
Classification of the modeling levels' data	**A**	**A:** accurate classification. **B:** Relative classification. **C:** lack of classification.
Support for abstraction levels (CIM, PIM, PSM)	**A**	**A:** abstraction levels and transitions are fully supported. **B:** All abstraction levels are defined, but transitions between them are not supported. **C:** some abstraction levels are not supported.
Structural, Behavioral, Functional modeling	**A**	**A:** All the system's aspects are modeled. **B:** some aspects of the system are not modeled.
Model Transformation type	**Both**	**Vertical:** The source and target models are at different levels of abstraction. **Horizontal:** The source and target models are at the same level of abstraction.
Automation level of transformations	**Medium**	**High:** Fully-automated. **Medium:** Semi-automated. **Low:** Manual.
Automatic code generation	**B**	**A:** All parts of the code are automatically generated. **B:** Most parts of the code are automatically generated. **C:** some parts of the code are automatically generated.
Tool support	**B**	**A:** A complete toolset is provided, or precise guidelines are defined to select alternative tools. **B:** A complete toolset is not provided, but general guidelines are provided to select alternative tools. **C:** No specific tools or guidelines are provided.
Round-trip engineering, Synchronization of source and target models, Verification/Validation	**B**	**A:** Detailed procedures are specified for the task in the methodology. **B:** Only general guidelines are provided for the task. **C:** The task is not covered by the methodology.

Table 4: Partial view of the results of assessing support for RESTful Web Services.

Criterion			Result	Status of *MDD4REST* based on the criterion
General Modeling of RESTful Web Services (Structural, Behavioral, Functional)			**A**	All aspects of modeling RESTful web services are considered.
RESTful Practices modeling (Resource, REST API Specification, Query, Third-Party API Integration)			**A**	All aspects relating to REST architecture are precisely defined. However, only guidelines are defined for third-party API Integration.
Security Modeling	Access Levels to Resources Access Levels to Resource Operations Definition of Roles		**A** **C** **A**	In Dynamic-security and Service models, access levels are defined for resources and roles, but access controls to resource operations are not supported.
Domain-Driven Modeling		Tactical Design Strategic Design	**A** **B**	The bounded context concept is not covered in *MDD4REST*. However, similar concepts, including application and module, are supported.
Web Engineering Architectures		Event-Driven Microservice Layered	**A** **A** **B**	In *MDD4REST*, the common Web architectures, including event-driven, microservices, and layered, are supported. The presentation layer must be completed by developers.
Process: CI/CD, First-API Design, Rich Domain Model			**A**	All of these activities are precisely defined in the process of *MDD4REST*.
Services: Service Integration and Service Granularity			**A**	The external services layer integrates internal services with third-party APIs. The Services are also divided into two layers: internal and external.
Operations: CRUD and Non-CRUD Operations, REST constraints, Error handling, Pagination, Filters			**A**	*MDD4REST* provides guidelines for all of these activities.
Test: Unit-Test, Integration-Test, API-Test, Test Case Generation			**B**	By default, the code generator will produce unit and API tests, but developers should also produce other types of tests.
Tools	Support for SQL and NoSQL Databases Support for Query Languages Use of Standard Technologies Diversity of Programming Languages		**A** **C** **A** **B**	MDD4REST uses popular code generators capable of supporting a wide range of products and programming languages.
Legend- A: Fully Supported**; B**: Partially supported; **C:** Not Supported**.**				

Table 5: Partial view of the results of assessing support for Model-Driven Web Engineering.

Criterion	Result	Status of *MDD4REST* based on the criterion
Existence of the Models Necessary for Web Development	**B**	Except for the presentation model, all the models necessary for designing web systems are supported.
Data Model (CIM, PIM, PSM)	**A**	Domain objects representing system data are obtained from the beginning in the form of ontologies and static models.
Business Model (CIM, PIM, PSM)	**A**	Business logic is provided in the form of Activity Storming models at the CIM level. At the following levels, this aspect can be seen in the Dynamic-Security and Service models.
Navigation Model (CIM, PIM, PSM)	**B**	Service navigation has been considered in the modeling levels of *MDD4REST*. However, presentation navigation has not been addressed, as the presentation aspect is outside the scope of *MDD4REST*.
Presentation Model (CIM, PIM, PSM)	**C**	In *MDD4REST*, the focus is on generating web services, and addressing the presentation aspect is outside the scope of *MDD4REST*.
Legend- A: Fully Supported; **B:** Partially supported; **C:** Not Supported.		

REFERENCES

Asadi, M., Esfahani, N., & Ramsin, R. (2010). Process patterns for MDA-based software development. In *ACIS International Conference on Software Engineering Research, Management and Applications*, 190–197.

Babanezhad, R., Bibalan, Y.M., & Ramsin, R. (2010). Process patterns for web engineering. In *IEEE Annual Computer Software and Applications Conference*, 477–486.

Blake, M. B. (2006). A lightweight software design process for web services workflows. In *IEEE International Conference on Web Services*, 411–418.

Brandolini, A. (2013). Introducing event storming. *blog, Ziobrando's Lair, 18*.

Deljouyi, A. (2021). MDD4REST Labs. Retrieved from https://github.com/MDD4REST

Ed-Douibi, H. (2019). *Model-driven round-trip engineering of REST APIs (PhD Thesis, Network and Information Technologies Doctoral Programme, Universitat Oberta de Catalunya)*.

Ed-Douibi, H., Izquierdo, J. L. C., Gómez, A., Tisi, M., & Cabot, J. (2015). EMF-REST: Generation of RESTful APIs from Models. In *Annual ACM Symposium on Applied Computing*, 1446–1453.

Evans, E. J., & Evans, E. (2004). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley Professional.

Fielding, R. T., & Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures* (Vol. 7). (PhD Thesis, University of California, Irvine).

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co.

Fowler, M. (2017). What do you mean by "Event-Driven"? Retrieved from https://martinfowler.com/articles/201701-event-driven.html

Gonçalves, R. C. da C., & Azevedo, I. (2018). *RESTful Web Services Development With a Model-Driven Engineering Approach*. (M.S. Thesis, Instituto Superior de Engenharia do Porto)

Greg Young. (2010). *CQRS Documents by Greg Young*.

Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, *45*(3), 451–461.

Haupt, F., Karastoyanova, D., Leymann, F., & Schroth, B. (2014). A model-driven approach for REST compliant services. In *International Conference on Web Services*, 129–136.

Hernandez-Mendez, A., Scholz, N., & Matthes, F. (2018). A Model-driven Approach for Generating RESTful Web Services in Single-Page Applications. In *International Conference on Model-Driven Engineering and Software Development*, 480–487.

Jegadeesan, H. (2009). *Towards a Model-Driven Approach to Support SOA-Based Web-Business Platforms*. (Ph.D. Thesis, BITS Pilani).

Kapferer, S. (2020). *A Modeling Framework for Strategic Domain-driven Design and Service Decomposition*. (M.S. Thesis, University of Applied Sciences of Eastern Switzerland).

Kolovos, D. S., Garcia-Dominguez, A., Rose, L. M., & Paige, R. F. (2017). Eugenia: Towards disciplined and automated development of GMF-based graphical model editors. *Software & Systems Modeling*, *16*(1), 229–255.

Koren, I., & Klamma, R. (2018). The Exploitation of OpenAPI Documentation for the Generation of Web Frontends. In *Web Conference*, 781–787.

Mulloy, B. (2013). *Web API design*. Academic Press.

Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media, Inc.

Ong, S. P. et al. (2015). The Materials Application Programming Interface (API): A simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles. *Computational Materials Science*, *97*, 209–215.

Rademacher, F., Sachweh, S., & Zündorf, A. (2017). Differences between model-driven development of service-oriented and microservice architecture. In *IEEE International Conference on Software Architecture Workshops*, 38–45.

Richardson, L., & Ruby, S. (2008). *RESTful web services*. O'Reilly Media, Inc.

Rodriguez, A. (2008). RESTful web services: The basics, *33*, 18 (Technical Report, IBM DeveloperWorks).

Schreier, S. (2011). Modeling RESTful applications. In *International Workshop on Restful Design*, 15–21.

Siegel, J. M. (2014). Model driven architecture (MDA)-MDA Guide rev. 2.0. (Technical Report, Object Management Group).

Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). *EMF: Eclipse Modeling Framework*. Pearson Education.

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., & Tsujii, J. (2012). BRAT: A web-based tool for NLP-assisted text annotation. In *Demonstrations Session at European Chapter of the Association for Computational Linguistics*.

Terzić, B., Dimitrieski, V., Kordić, S., Milosavljević, G., & Luković, I. (2018). Development and evaluation of MicroBuilder: A Model-Driven tool for the specification of REST Microservice Software Architectures. *Enterprise Information Systems*, *12*(8–9), 1034–1057.

Truyen, F. (2006). The fast guide to model driven architecture. *Cephas Consulting Corp*.

Valverde, F., & Pastor, O. (2009). Dealing with REST services in model-driven web engineering methods. *V Jornadas Cientifico-Técnicas en Servicios Web y SOA*, 243–250.

Zolotas, C., Diamantopoulos, T., Chatzidimitriou, K. C., & Symeonidis, A. L. (2017). From requirements to source code: A Model-Driven Engineering approach for RESTful web services. *Automated Software Engineering*, *24*(4), 791–838