Co-Design of Business and IT Services - a Tool-Supported Approach

Blagovesta Pirelli[0000−0001−9890−5227], Natalia Nessler, and Alain Wegmann

Ecole Polytechnique Fédérale de Lausanne (EPFL),
School of Computer and Communication Sciences
CH-1015 Lausanne, Switzerland
{blagovesta.pirelli, natalia.nessler, alain.wegmann}@epfl.ch

Abstract. Service modeling is an important step in designing service-oriented systems. There are multiple levels of design because service science includes both the business rationale and the IT implementation of the services. As business and IT perspectives differ, the modeling techniques are different, and often the respective modeling languages are disconnected or ad-hoc. We propose a new service-modeling approach for connecting the business modeling and the web service modeling by presenting these two perspectives in a single model. We present a multi-stage modeling process for capturing different perspectives and creating models iteratively by working with levels of abstraction from higher to lower. The model is then used as an input in order to generate a REST API specification in the OpenAPI format to feed the next stages of the service life-cycle.

Keywords: Service description · Service modeling · Service specification · OpenAPI · REST

1 Introduction

In today's API economy, many business-information systems make use of web services or develop interfaces for other systems to interact with them [14]. During the development of these systems, a major part of the work is the definition and configuration of the web services. The business requirements for these services are often captured with user stories, or other informal or semi-formal descriptions. However, the development of the actual services requires a fine level of detail.

We use SEAM [15], a modeling technique based on service science, to bring together abstract business models and precise specifications. SEAM enables us to understand the business environment and to define the actors responsibilities for the information managed by them. We observe that service-modeling techniques fall into two categories: business-service modeling and IT-service modeling. On the one hand, the business-service models are not precise enough to fully describe web services (e.g., which identifiers to use, which status to return to the user). On the other hand, IT-service models do not preserve the business semantics.

Therefore, our research question is: *How do we extend existing service modeling to simultaneously design business and IT services?*

We present a service-modeling approach for generating web-service specifications by using an ontological extension of service models that captures the minimum amount of annotations necessary to define the web services. With these annotations, a single model captures all business and technical requirements, and enables hiding or showing information as necessary. For web services, we chose the Representational State Transfer (REST) [6], the widely adopted architectural style for web services. There are many languages that describe REST services, e.g., WADL [7]. We chose to work with the OpenAPI[1] specification (previously Swagger), as it is the *de facto* industry standard. We developed a supporting tool that generates the OpenAPI specification corresponding to the service models. Our tool removes the necessity of transferring the business models to technical specifications and is a part of the toolbox for service designing with SEAM.

Using our proposed modeling approach, a service designer (i.e., a business analyst, a requirements engineering practitioner, or a project manager) goes through the following process. First, the service designer captures the business environment in a service-system model. Then, they describe the information entities that the business actors use in the service process. Next, they annotate the models with additional information necessary for describing IT services that each actor requires. Finally, our tool generates the web specification from the models.

This paper's structure is the following: In Section 2, we present a brief literature review and background work. In Section 3, we describe our approach for the generation of REST API specifications from service models. In Section 4, we conclude and outline future work.

2 Literature

2.1 Services in Business and IT

A service system is a set of elements that collaborate in a service delivery process [8]. A service system is most often a socio-technical system that presents the interaction between both human and technical elements. Modeling complex systems, with the help of conceptualization and formalization of the target service system, is a necessary step for understanding their behavior. Service models show only an abstraction, i.e., abstract away any underlying implementations. With simplifications such as modularization, abstraction, and interfaces, services are defined in a minimalistic way, and understandable by the people to whom service designers communicate the models.

One of the earliest works on service modeling, service blueprinting [13], lays the four fundamental steps to modeling and designing services: (1) identify processes, (2) isolate fail points, (3) timebox the execution, and (4) analyze profitability. The service blueprint is generic and does not take into account the recent developments of the IT infrastructure involved in the service process because

[1] `https://www.openapis.org/`

it was developed in the 80's. Recently, Estaol et al. [5] used service blueprinting to extend service modeling and designing to a set of executable logic rules to check the validity of the service blueprint.

For specialized modeling of business processes within an organization, Business Process Modeling and Notation (BPMN) is widely used. BPMN's logic is similar to an activity diagram, which makes the notation easy to understand. The models can be executed (if complete). However, BPMN does not tolerate ambiguity that is present at the beginning of service design.

IT-service modeling techniques include mostly formal description languages in order to specify the behavior of web services. For example, WSDL [7] and the OpenAPI are such specifications. The resulting specifications are highly technical and detailed, and they do not accommodate a more abstract view of who uses the services for which purpose. Nevertheless, these formal specifications are necessary for developing the service according to its life-cycle steps [12].

Recently, web-service verification has been a subject of interest to researchers as it is a means to prove that services implementation complies with the rules and policies that service designers define at a business level. It is still a non-trivial task to verify distributed systems without a coordination module (a gateway in traditional SOA terms), but the advancements of formal methods are promising even if still incomplete and costly, as shown by Camilli et al. [2] and Panda et al. [11]. For such verification, a formal specification is a necessary but insufficient condition. The verification models need to include the information from the context, as well as the local conditions of the service state. Complex scenarios that require business logic and rules co-exist with the web service descriptions.

In this paper, we use SEAM, a service science modeling method, for the basis of our work. SEAM models show different levels of abstraction by instantiating a service model with only one single perspective. The type of information and the level of detail of a service model depend on whether the model is meant for business people or for engineers. A SEAM diagram shows these perspectives (levels) in a hierarchy of systems [15]. The typical levels of an enterprise are a business segment, an organization, a department/team, and an IT system (infrastructure). These levels are refined in the same way as an interface (black box) is a refinement of the module that implements the interface (white box). In the same way, service system models form a hierarchy of refined service system models.

The SEAM method includes service behavioral models that capture the service system's actors and the relationships between them. A behavioral model (illustrated in Figure 1) contains working objects (either business- or human-working objects) and relationships between them. The models are hierarchical: a working object as a whole (noted with "[w]") hides its implementation details from other working objects. With the refinement relationship between working objects, we relate and refine working objects as a whole to the working object as a composite (noted with "[c]"), and we "see" how the service system is organized. Service-system refinement means that a working object as a whole is an

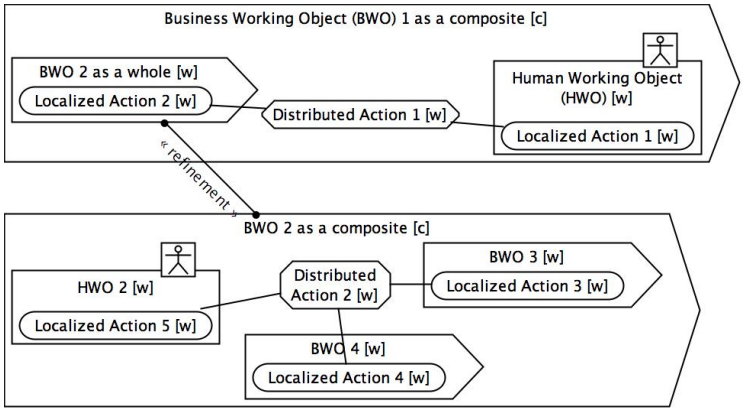

Fig. 1. Service System's Graphical Notation

abstraction of a working object as a composite; and vice-versa, a working object as a composite is a refinement for a working object as a whole.

2.2 Web Services Generation

The generation of REST APIs from different sources is an active research domain. Recent work in the area includes projects that show how to generate an OpenAPI specification from HTML documentation [3] or how to extract automatically REST specifications from deployed web sites or from code analysis [4]. These generative tools rely on already existing web resources and do not connect the business services to the web services.

The semantic annotation of web services has yielded many results, most notably OWL-S [9,10]. However, OWL-S is a bottom-up approach. It is based on WSDL and requires a level of technical detail unfeasible to achieve at the business-service level.

There are other methods in the domain of service-oriented software development, for example, the SOMA method [1] for building service-oriented solutions. SOMA uses different models to represent business and IT requirements for the specification of services. Furthermore, there is no automation of the specification process, which makes it costly to develop, update, and coordinate the models corresponding to business and the models corresponding to IT services.

2.3 Summary

None of the existing service-design and modeling methods binds together the rigor of formal methods and the vigor of the service-design process. The business environment poses constraints (e.g., policies, business rules, semantics) to the implementation; these constraints are hard to maintain or to express formally because of the different ways of modeling human cognition (natural language,

informal formulation) and of IT services (formal syntax and semantics of web service definition languages and specifications). Generative tools are useful for avoiding mistakes by carrying out repetitive, mechanical actions. These tools also ease the subsequent design steps but do not substitute human input in the design process. As design is iterative, at each step, system designers need to provide constraints (input, outputs, and invariants) and implementation details.

Instead of abstracting all details of the IT implementation from the business perspective, we add to the business service models a few annotations for web services descriptions and provide a tool in order to automate model translations. All details of the requirements of the web-service specification are captured where the benefits and the use of the web service are captured. Not all web services are specified from the business-service model. Infrastructural services and other utilities are not visible at the business level of abstraction. However, the generated service specification would suffice as an input for initiating the web-service contract design and the web-service implementation.

3 Proposed Modeling Method

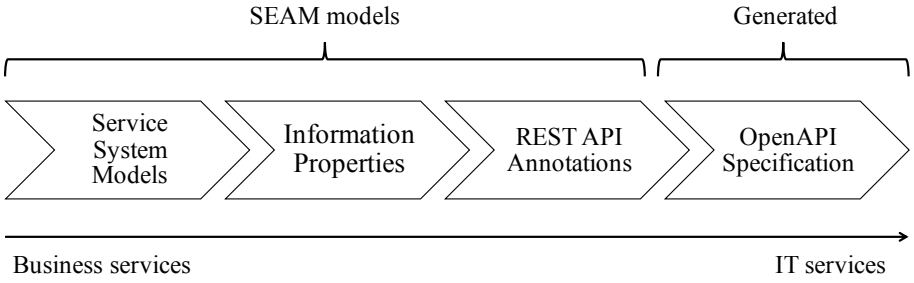

Fig. 2. Co-design modeling approach for business and IT services

Recall that our proposed modeling method is based on SEAM, a service design method [15]. Our proposed method includes the following four modeling steps (Fig. 2):

1. Business-service modeling – the service designer creates the high-level business-service system models
2. Information-properties definition – the service designer includes the information that each actor operates on; this information represents the data entities on which a web service operates as well
3. REST annotations – the designer includes the annotations necessary for the specification of RESTful web services
4. IT-services specification – our tool generates a concrete REST API specification in the form of an OpenAPI specification

The first three steps require the service designer to model the services: observe the environment, conceptualize their observations in the form of models, and/or design new service systems for prototyping. During these steps, the service designer uses a CAD tool to create formal service models. The last step is where our tool generates the web service specification. The modeling method is iterative in its nature and helps service designers quickly develop minimal service-specifications for both business and IT.

We use a running example to explain the modeling steps in detail. The example we model is the maintenance service for airplanes with a particular type of engine. The example models show how an airplane club receives the service provided by the value network for maintaining airplanes with an engine manufacturer's (EM) engine.

3.1 Step 1: Service System Model

To model both the business service and the corresponding REST API, service designers analyze the environment and prepare the service system models of the business case. The business-service system includes the actors involved in the service process, the services they provide, and the process in which they interact. The service designer models the service system with the help of a CAD tool that generates both an XML meta model and a graphical representation of a service behavioral model.

Business-Service Model of the Interaction with the Client The service designer models the service model of the service provider and their clients (or service adopters). Figure 3 depicts the service system called *airplane maintenance*. There are two actors: a business working-object named *engine manufacturer's (EM) value network* and a human working-object named *airplane club*. The process in which they interact is the *airplane maintenance* process. The *engine workshop* provides the service *fix an airplane with an EM's engine*.

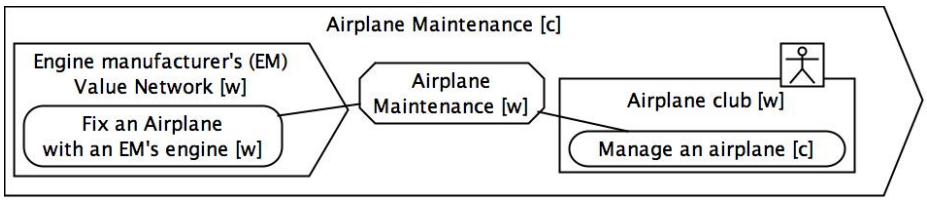

Fig. 3. Business Service Model of the Interaction with the Client

Business Service Model of the Interaction with the Client and the Refined Value Network of the Provider Next, the service model includes

the details about who is a part of the value network that delivers the service *fix an airplane with an EM's engine*. The details are depicted in Figure 4. The model is refined with the service system as a composite of the *EM's value network*. The composite service system includes *EM*, a *certified technician*, and an *engine repair shop*. The value network disregards organizational boundaries and includes all actors who collaborate in a service process based on the service they provide.

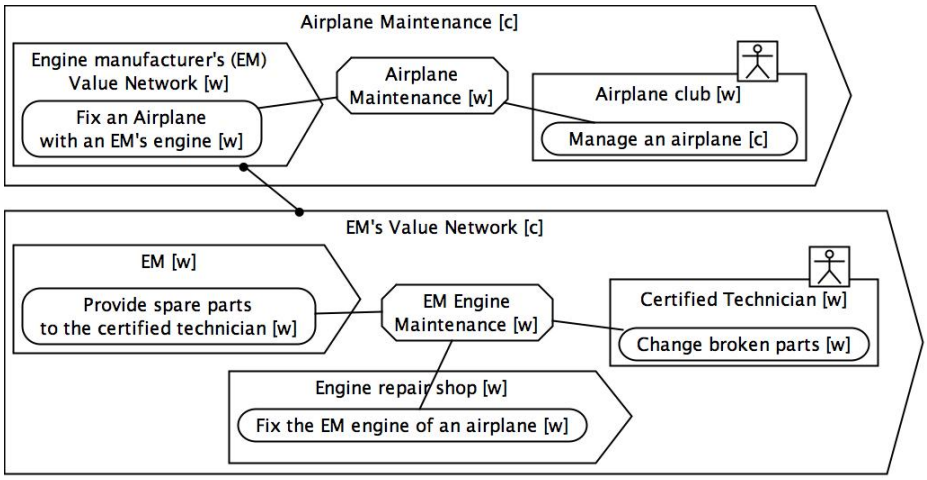

Fig. 4. Business Service Model of the Interaction with the Client and the Refined Value Network of the Provider

3.2 Step 2: Information Properties

An important step in understanding the context of the service is to define the concepts related to the service process. Different actors have different vocabularies. A classic way of looking at these concepts is with the help of entity-relationship diagrams and domain-specific languages (DSL). Our models capture the information necessary for providing a service with its localized properties.

In the case of the plane maintenance service-system, we define the information properties of the services of each working object (Figure 5). On the level the services related to airplane maintenance, the vocabulary of the actors includes information properties for *Airplane*, *Repair*, and *Repair of Airplanes*. In the model, the airplane club has both *airplane*s and *repair*s, as they manage airplanes that sometimes break and need repairs. The EM's information includes only *repair of airplanes* because they are not concerned with the airplanes, only with their repair. The refined model includes actors that manage the information related to airplanes and airplane parts (e.g., *airplane*, *engine*, *part*), and that have a vocabulary related to handling broken parts (e.g., *certification*).

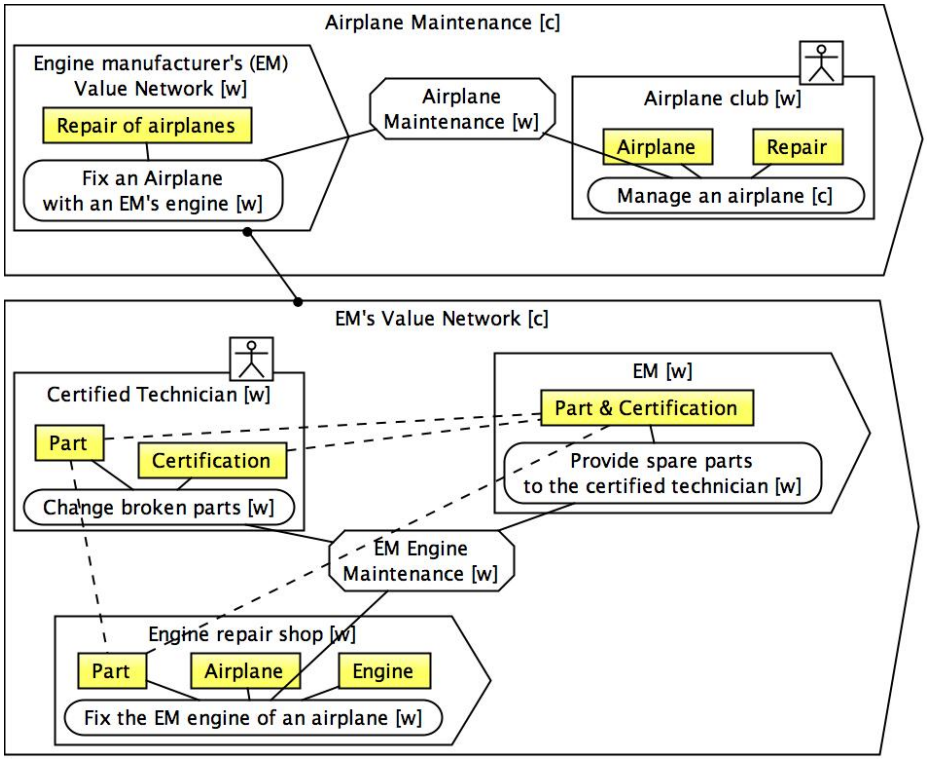

Fig. 5. Information Properties

The *part* and *Certification* information properties are connected with dashed lines to show the relationship between the concepts for different actors. The *engine repair shop* has the information on the broken part (which belongs to an engine, hence, to an airplane). This information initiates other information flows; *EM* now knows which part has to be shipped to the technician who has the appropriate level of *certification* to change the broken part. In this service interaction, the status attributes of the *part* are transformed, but the *part* remains the same information property.

3.3 Web Services Annotation

After the information properties are defined, the service model includes a minimal annotation for describing the web services necessary to operate with this information. We create our annotations by using the fact that there are only four required properties to create a valid OpenAPI specification. These properties are a base URL, the path templates, the HTTP verbs, and the formal parameters.

Inside a localized action, the SEAM meta model includes localized actions that correspond to web-service operations. Thus, we include the HTTP verb,

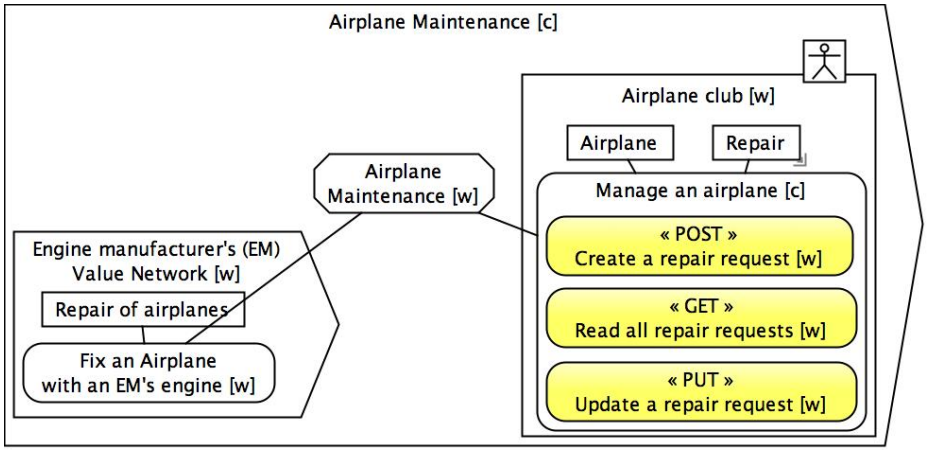

Fig. 6. Web services operations annotations

e.g., POST, GET, PUT, DELETE, in the localized action as the additional XML property *stereotype*. In Figure 6, the *manage an airplane* localized action includes three sub-localized actions that correspond to managing the information known to the *airplane club*. These three sub-localized actions are (1) <<POST>> Create a repair request, (2) <<GET>> Read all repair requests, and (3) <<PUT>> Update a repair request. They can be automatically generated to include a sub-localized action for each pair *<HTTP verb, Information Property>*.

Furthermore, the sub-localized actions include the rest of the required web-service description parts: the parameters and the path. In Figure 7, the parameters are included as localized properties, annotated semantically with a *stereotype* property *<<in>>* for input parameters and *<<out>>* for output parameters. The path template is described by a sequence edge which points from the input to the output parameter and is labeled with the path template. The parameters have a predefined syntax for describing what type of object the parameter assumes (a built-in type in the OpenAPI, e.g., `string`, `integer`, etc., or a complex type, e.g., `enum`, `array`, `schema`). We give examples for possible input and output parameters in Listing 1.1 for `built-in` parameters, in Listing 1.2 for `schema` parameters, in Listing 1.3 for `enum` parameters, and in Listing 1.4 for `array` parameters.

Listing 1.1. Built-in type parameters example

```
name: string ,
age: integer
```

Listing 1.2. Schema parameter example

```
person {
        name: string ,
        age: integer
}
```

Listing 1.3. Enumeration parameter example

```
count: enum(one, two, three)
```

Listing 1.4. Array parameters example

```
people: array(string),
students: array(person),
profs: array(person {name: string, age: integer})
```

Figure 7 shows the syntax for the three services defined for the airplane club. For example, *<<POST>> Create a repair request* expects a `NewRepair` object with attributes `clientId, planeId, date, description` as an input parameter. The output parameter, which the web service sends after execution, is a `Repair` object with attributes `repairId` and a `status` of an *enum* type and values `new, wip, done`.h

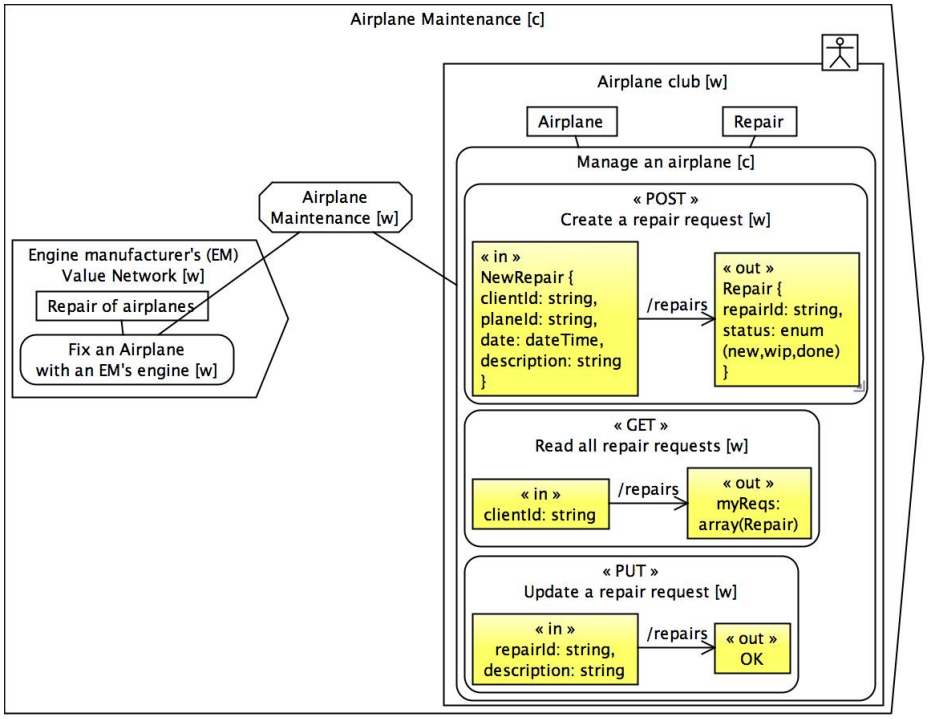

Fig. 7. Web service's parameters annotation

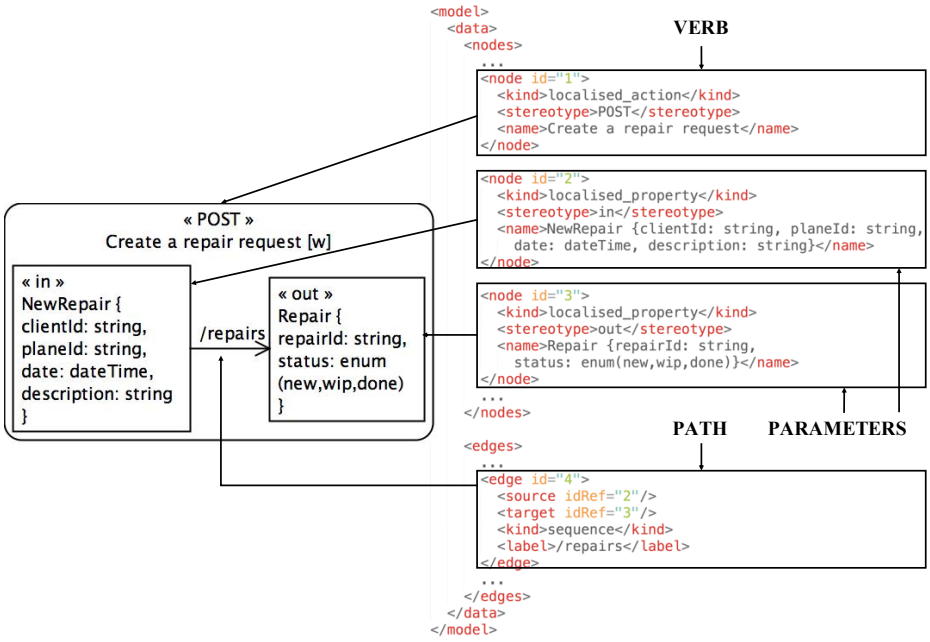

Fig. 8. Meta model annotations for web services

3.4 OpenAPI specification

We developed a tool[2] that takes as an input the XML file of the service model and generates the OpenAPI specification from it. During our research, we created new annotations for our models. Figure 8 shows the relationship between the graphical model and the meta model for these annotations. In our tool, we use the standard SEAM model without changing it.

There are only two types of elements in the SEAM meta model: `nodes` and `edges`. Nodes represent elements and edges represent relationships between elements. Nodes have a name, an ID, a stereotype, a kind, and other attributes. Edges have an ID, a source, a target, a label, a kind, and other attributes. We use only existing attributes to store the information from the web-service annotations. Existing tools can still parse the models but do not give semantic meaning to the new annotations stored in the attributes. Our tool, when it parses the model, assumes a semantic meaning for creating the web-service specifications for these attributes.

Within every `node` of type `localized_action` that requires a web service, we add one or many sub-`localized_action`s. The `node` then has a property `stereotype` that defines the `verb` of the web service. Within the `localized_action` node, there are two other `nodes`. The `edge` defining which `node` includes which other `nodes` is not in the figure. The two nodes of the same type of

[2] `https://github.com/lams-epfl/gen-rest/`

`localized_property` also include a `stereotype` property. The `stereotype` of these two `nodes` is either `in` or `out` in order to show which parameter the `node` describes. Our tool interprets the property `stereotype` of a `node`, based on the property `kind` of the type of the `node`, either a `localized_property` or a `localized_action`). The last piece of information is the `path` of the web service. The path is described by the `edge` that connects the input and the output parameters. The property `label` of the `edge` contains the path template.

The final result from our tool is a correct runnable OpenAPI specification in the YAML data format and can be used to continue the web-service life-cycle. Listing 1.5 depicts the generated output for our example.

Listing 1.5. Generated OpenAPI 3.0.0 YAML

```yaml
openapi: 3.0.0
servers: []
info:
  version: 1.0.0
  title: Airplane maintenance
tags:
  - name: EM VN
  - name: Airplane club
  - name: Technician
  - name: Engine repair shop
  - name: EM
paths:
  /repairs:
    post:
      tags:
        - Airplane club
      description: Create a repair request
      responses:
        '200':
          description: request successful
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Repair'
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/NewRepair'
        required: true
    get:
      tags:
        - Airplane club
      description: Read all repair requests
      parameters:
        - name: clientId
          in: query
          required: true
          schema:
            type: string
      responses:
        '200':
          description: request successful
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Repair'
    put:
      tags:
```

```
        - Airplane club
      description: Update a repair request
      responses:
        '200':
          description: OK
      requestBody:
        content:
          application/json:
            schema:
                type: object
                properties:
                  clientId:
                    type: string
                  description:
                    type: string
        required: true
components:
  schemas:
    NewRepair:
      type: object
      properties:
        clientId:
          type: string
        planeId:
          type: string
        date:
          type: string
          format: date time
        description:
          type: string
    Repair:
      type: object
      properties:
        repairId:
          type: string
        status:
          type: string
          enum:
            - new
            - wip
            - done
```

4 Conclusion and Future Work

In this paper, we have proposed a method how to co-design business-service systems and their corresponding REST API specifications. Our method consists of four modeling steps: (1) modeling of the business service system, (2) conceptualization of domain information; (3) annotation of the service system model with attributes describing RESTful services, and (4) generation of the REST specification in the form of an OpenAPI specification.

The next steps of the project are to develop a bidirectional connection between models and to automate the generation of SEAM models from an OpenAPI specification. Despite the fact that bringing semantics to web-service annotations is an already established research area, our approach differs because it brings these annotations to the business-model level.

Moreover, we will work on model checkers and parsers that ensure correctness of the models. The first step would be to parse the OpenAPI definition in order to propose to the CAD users what they could use when developing

the services. Our long-term goal is to integrate support for REST APIs that have been already developed and published in order to help designers with the modeling process when there is no need for a web-service implementation but a web-service configuration.

References

1. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: Soma: A method for developing service-oriented solutions. IBM systems Journal **47**(3), 377–396 (2008)
2. Camilli, M., Bellettini, C., Capra, L., Monga, M.: A formal framework for specifying and verifying microservices based process flows. In: International Conference on Software Engineering and Formal Methods. pp. 187–202. Springer (2017)
3. Cao, H., Falleri, J.R., Blanc, X.: Automated generation of rest api specification from plain html documentation. In: International Conference on Service-Oriented Computing. pp. 453–461. Springer (2017)
4. Choudhary, S., Kimura, K., Sekiguchi, A.: Spec2rest: An approach for eliciting web api resources from existing applications. In: Web Services (ICWS), 2017 IEEE International Conference on. pp. 910–913. IEEE (2017)
5. Estañol, M., Marcos, E., Oriol, X., Pérez, F.J., Teniente, E., Vara, J.M.: Validation of service blueprint models by means of formal simulation techniques. In: International Conference on Service-Oriented Computing. pp. 80–95. Springer (2017)
6. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT) **2**(2), 115–150 (2002)
7. Hadley, M.J.: Web application description language (wadl). Tech. rep., Mountain View, CA, USA (2006)
8. Maglio, P.P., Vargo, S.L., Caswell, N., Spohrer, J.: The service system is the basic abstraction of service science. Information Systems and e-business Management **7**(4), 395–406 (2009)
9. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al.: Owl-s: Semantic markup for web services. W3C member submission **22**(4) (2004)
10. Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., et al.: Bringing semantics to web services: The owl-s approach. In: International Workshop on Semantic Web Services and Web Process Composition. pp. 26–42. Springer (2004)
11. Panda, A., Sagiv, M., Shenker, S.: Verification in the age of microservices. In: Proceedings of the 16th Workshop on Hot Topics in Operating Systems. pp. 30–36. ACM (2017)
12. Papazoglou, M.P., Van Den Heuvel, W.J.: Service-oriented design and development methodology. International Journal of Web Engineering and Technology **2**(4), 412–442 (2006)
13. Shostack, G.L.: Designing services that deliver. Harvard Business Review **62**(1), 133–139 (1984)
14. Tan, W., Fan, Y., Ghoneim, A., Hossain, M.A., Dustdar, S.: From the service-oriented architecture to the web api economy. IEEE Internet Computing **20**(4), 64–68 (2016)
15. Wegmann, A.: On the systemic enterprise architecture methodology (SEAM). In: ICEIS 2003, Proceedings of the 5th International Conference on Enterprise Information Systems, Angers, France, April 22-26, 2003. pp. 483–490 (2003)