Towards Transforming OpenAPI Specified Web Services into Planning Domain Definition Language Actions for Automatic Web Service Composition

Christian Schindler, Christoph Knieke,
Andreas Rausch, and Eric Douglas Nyakam Chiadjeu

Technische Universität Clausthal
Institute for Software and Systems Engineering
Clausthal-Zellerfeld, Germany

The Fifteenth International Conference on Adaptive and Self-Adaptive Systems and Applications ADAPTIVE 2023

TU Clausthal

ISSE Institut für Software and Systems Engineering

About the presenter

- Christian Schindler received a master‘s degree in business informatics from the university of Mannheim, Germany in 2016. He is currently a doctoral researcher at the Institute for Software and Systems Engineering at Technische Universität Clausthal, Germany.

- Research interest lies in software engineering, software architecture and inductive rule learning

- Our group (Data-based Software Engineering Methods and Tools) is interested in utilizing all kinds of data along the lifecycle of complex systems (development artifacts, models, runtime traces …) to support the engineering process.
- We develop methods and tools from supporting decision-making up to automating parts of the engineering process.

Goal

- Enabling ad-hoc composition of available web service descriptions without the need to explicitly configure all possible combinations upfront
  - flexibly compose web services on-the-fly based on user specific requirements
  - seamless integration of different web service descriptions by automatically handling the composition process
- Preparing the self-adaptive composition mechanism
  - dynamically adjust and optimize the composition of web services based on changing conditions
    - Previously unseen requirements
    - (un)available service descriptions

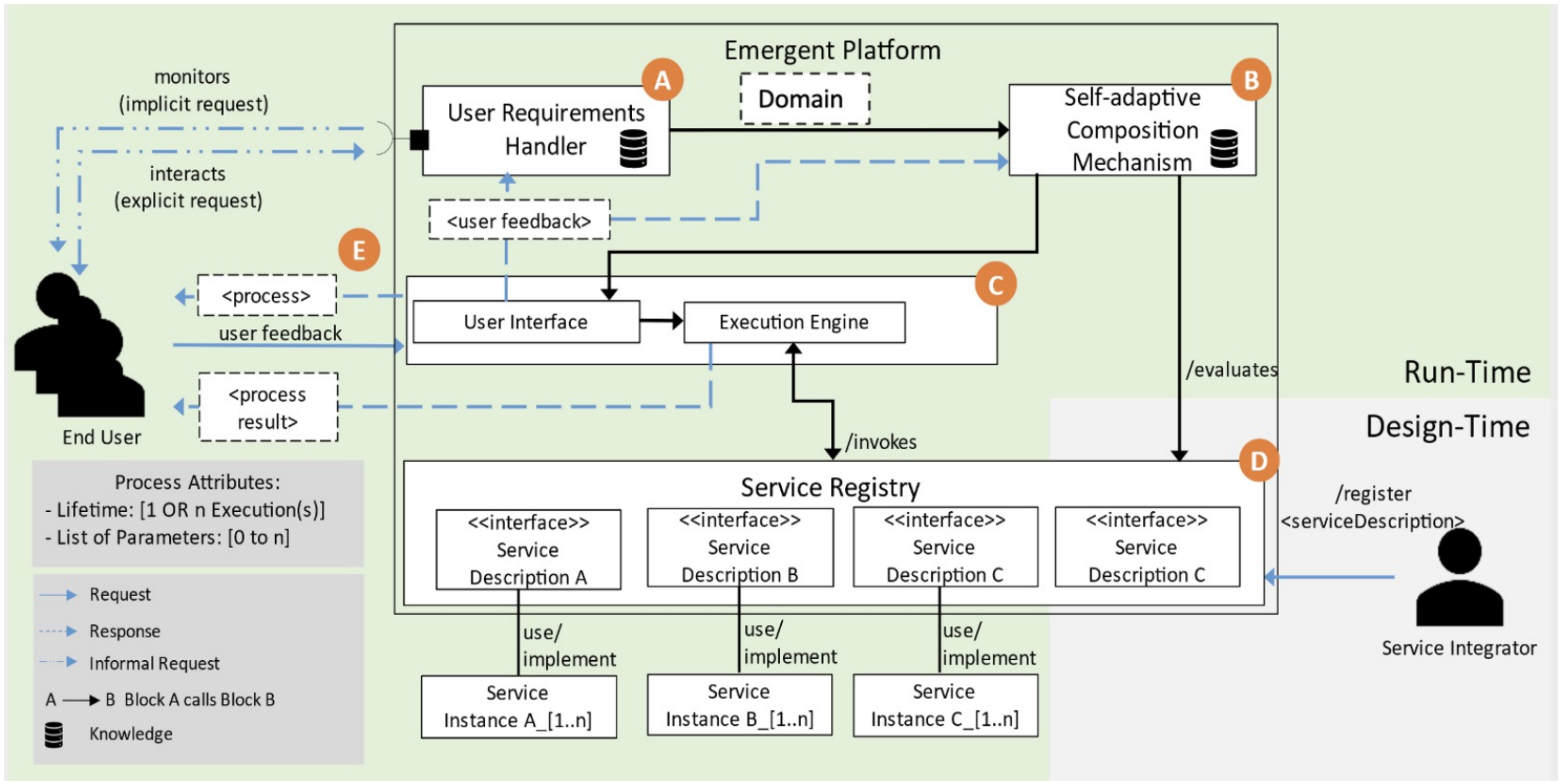
Contribution

- Developed **a set of rules** for transforming web service specifications (OpenAPI) into Planning Domain Definition Language (PDDL) actions and domains to enable composition and meet higher-level requirements within the overall platform.
- Creation of actions (with parameters, preconditions, and effects) and the corresponding domain from OpenAPI specifications and the underlying data schema.

TU Clausthal

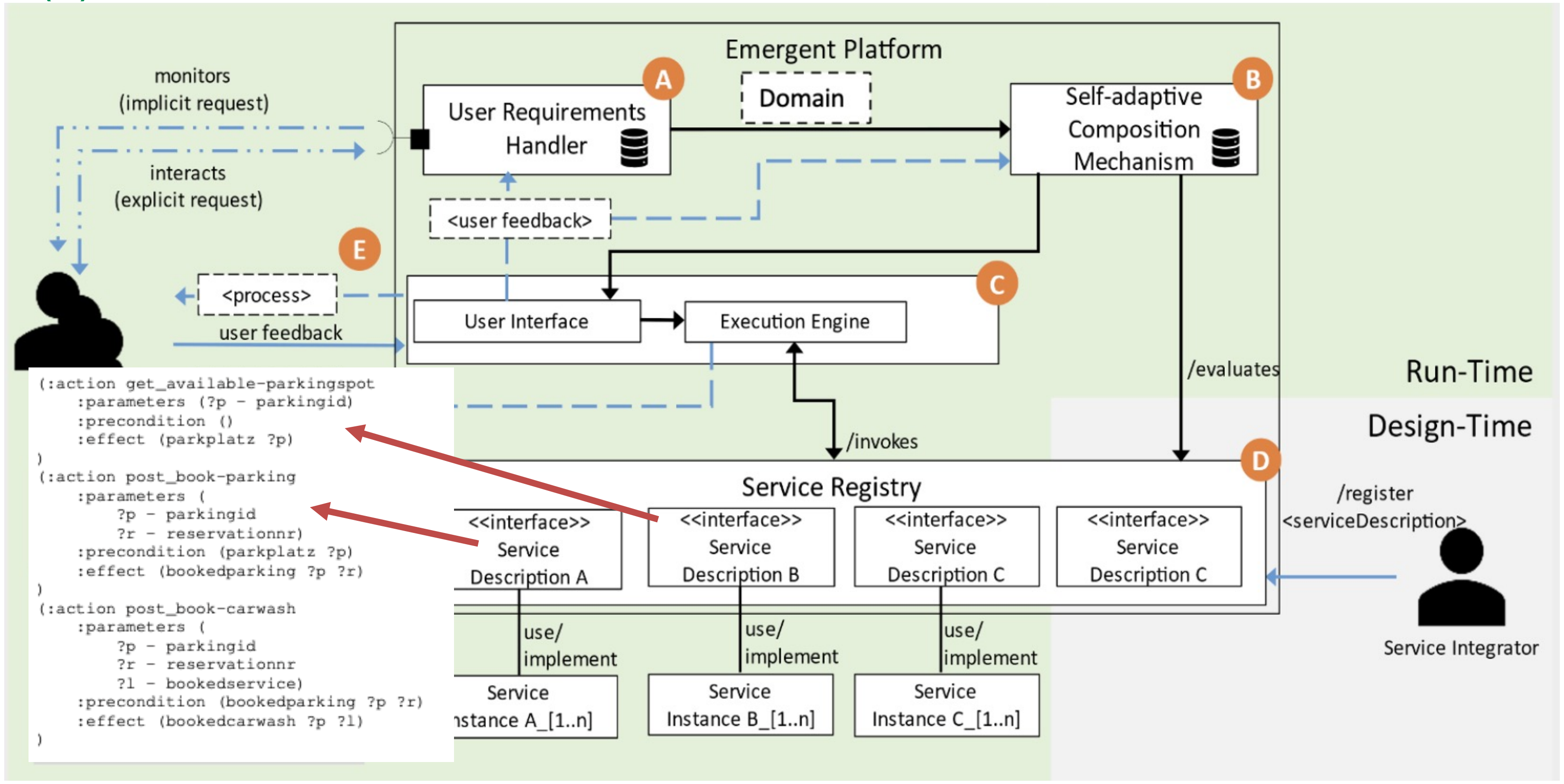
Institut für Software and Systems Engineering

Context: Platformarchitecture



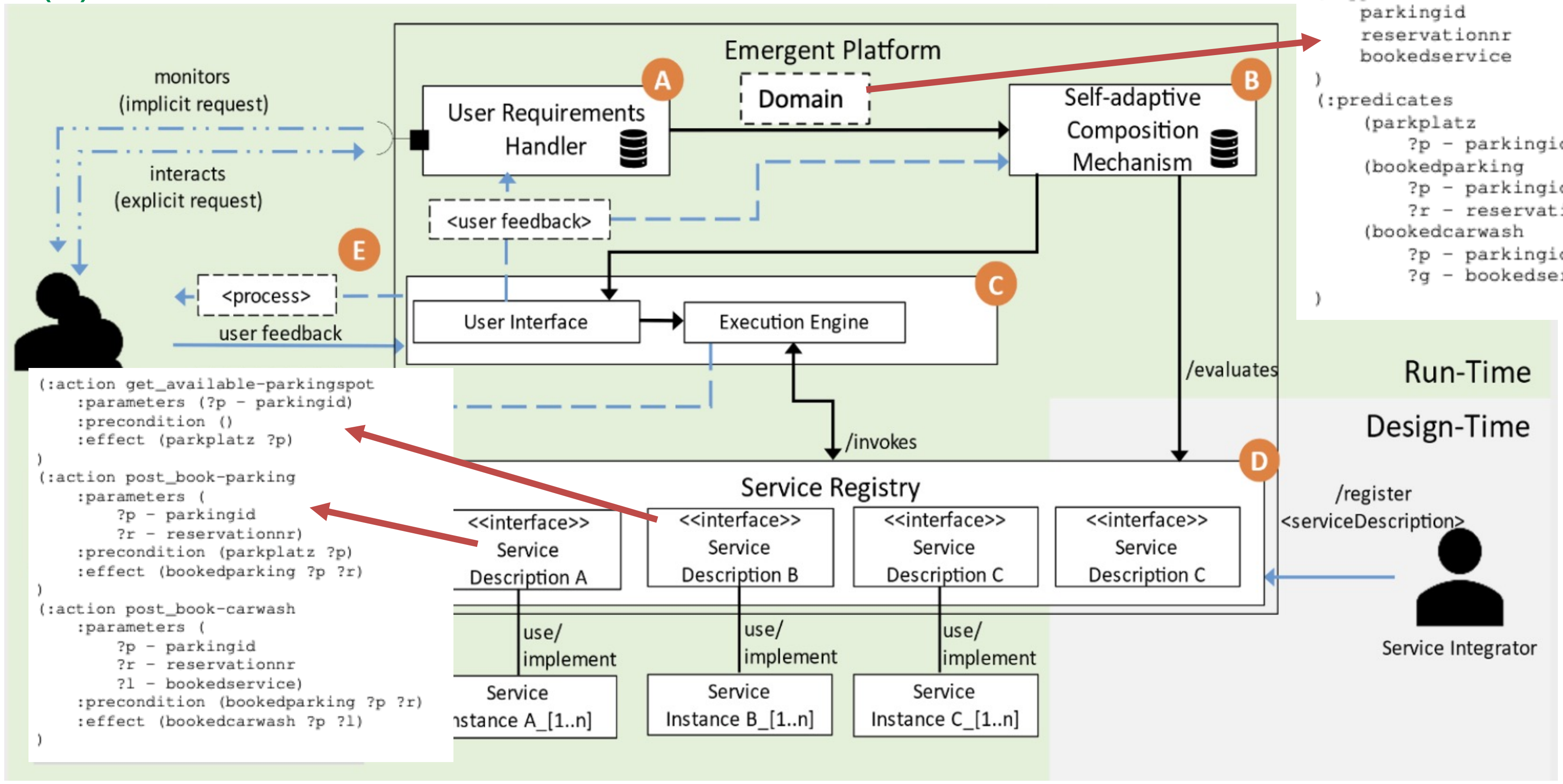
Emergent Platform
Domain
A
User Requirements Handler
B
Self-adaptive Composition Mechanism
monitors (implicit request)
interacts (explicit request)
<user feedback>
E
<process>
user feedback
<process result>
End User
C
User Interface
Execution Engine
/evaluates
/invokes
Run-Time
Design-Time
D
Service Registry
<<interface>> Service Description A
<<interface>> Service Description B
<<interface>> Service Description C
<<interface>> Service Description C
use/ implement
Service Instance A_[1..n]
Service Instance B_[1..n]
Service Instance C_[1..n]
/register <serviceDescription>
Service Integrator
Process Attributes:
- Lifetime: [1 OR n Execution(s)]
- List of Parameters: [0 to n]
Request
Response
Informal Request
A → B Block A calls Block B
Knowledge

Motivating example for the PDDL based description to enable composition of web service (1)



```
(:action get_available-parkingspot
    :parameters (?p - parkingid)
    :precondition ()
    :effect (parkplatz ?p)
)
(:action post_book-parking
    :parameters (
        ?p - parkingid
        ?r - reservationnr)
    :precondition (parkplatz ?p)
    :effect (bookedparking ?p ?r)
)
(:action post_book-carwash
    :parameters (
        ?p - parkingid
        ?r - reservationnr
        ?l - bookedservice)
    :precondition (bookedparking ?p ?r)
    :effect (bookedcarwash ?p ?l)
)
```

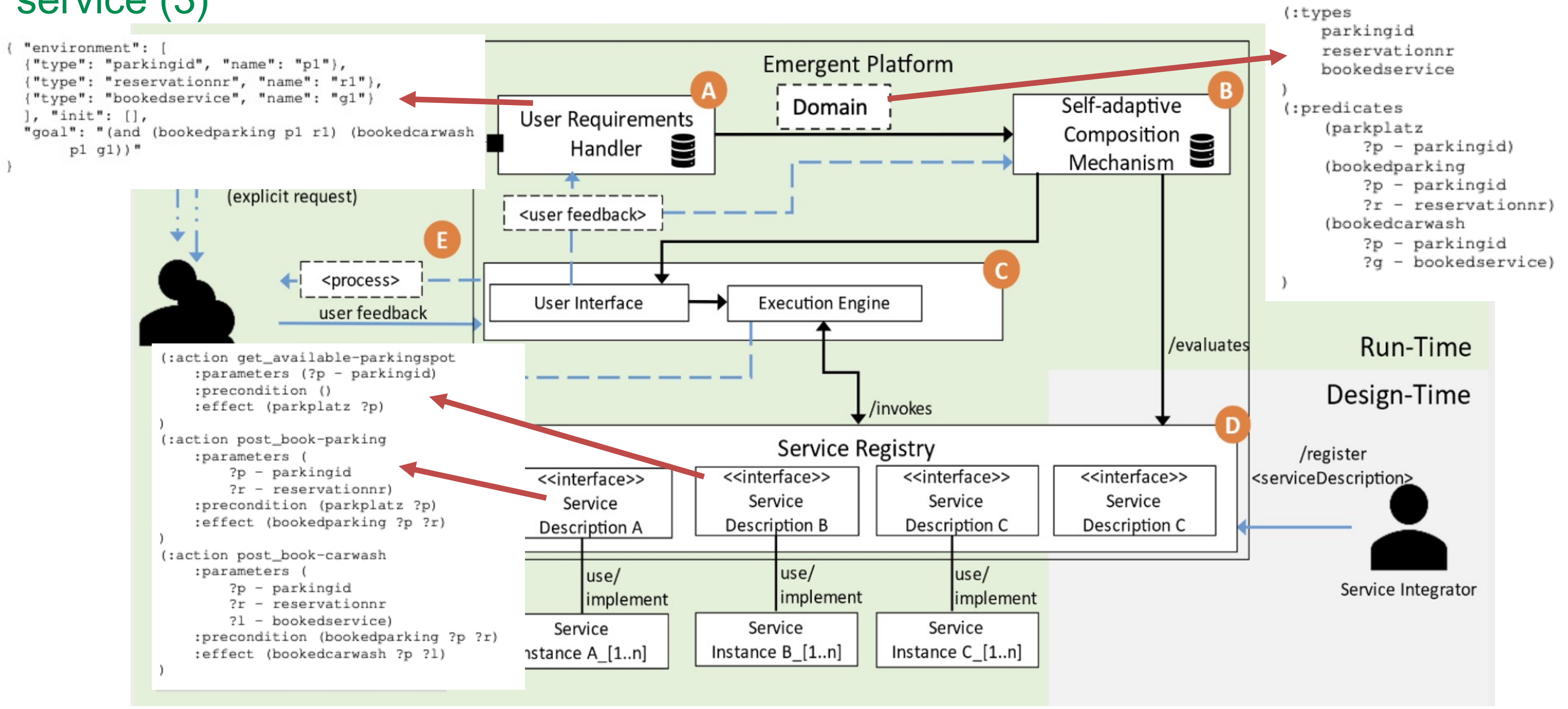
Motivating example for the PDDL based description to enable composition of web service (2)


Emergent Platform
monitors (implicit request)
interacts (explicit request)
A
User Requirements Handler
Domain
B
Self-adaptive Composition Mechanism
<user feedback>
E
<process>
user feedback
C
User Interface
Execution Engine
/evaluates
Run-Time
Design-Time
/invokes
D
Service Registry
<<interface>> Service Description A
<<interface>> Service Description B
<<interface>> Service Description C
<<interface>> Service Description C
/register <serviceDescription>
use/ implement
use/ implement
use/ implement
Service Instance A_[1..n]
Service Instance B_[1..n]
Service Instance C_[1..n]
Service Integrator

```
(:types
    parkingid
    reservationnr
    bookedservice
)
(:predicates
    (parkplatz
        ?p - parkingid)
    (bookedparking
        ?p - parkingid
        ?r - reservationnr)
    (bookedcarwash
        ?p - parkingid
        ?g - bookedservice)
)
```

```
(:action get_available-parkingspot
    :parameters (?p - parkingid)
    :precondition ()
    :effect (parkplatz ?p)
)
(:action post_book-parking
    :parameters (
        ?p - parkingid
        ?r - reservationnr)
    :precondition (parkplatz ?p)
    :effect (bookedparking ?p ?r)
)
(:action post_book-carwash
    :parameters (
        ?p - parkingid
        ?r - reservationnr
        ?l - bookedservice)
    :precondition (bookedparking ?p ?r)
    :effect (bookedcarwash ?p ?l)
)
```

Motivating example for the PDDL based description to enable composition of web service (3)


Emergent Platform
Domain
A
User Requirements Handler
B
Self-adaptive Composition Mechanism
(explicit request)
<user feedback>
E
<process>
user feedback
C
User Interface
Execution Engine
/evaluates
Run-Time
Design-Time
/invokes
D
Service Registry
<<interface>> Service Description A
<<interface>> Service Description B
<<interface>> Service Description C
<<interface>> Service Description C
/register <serviceDescription>
Service Integrator
use/ implement
use/ implement
use/ implement
Service Instance A_[1..n]
Service Instance B_[1..n]
Service Instance C_[1..n]

```
{ "environment": [
  {"type": "parkingid", "name": "p1"},
  {"type": "reservationnr", "name": "r1"},
  {"type": "bookedservice", "name": "g1"}
  ], "init": [],
  "goal": "(and (bookedparking p1 r1) (bookedcarwash
       p1 g1))"
}
```

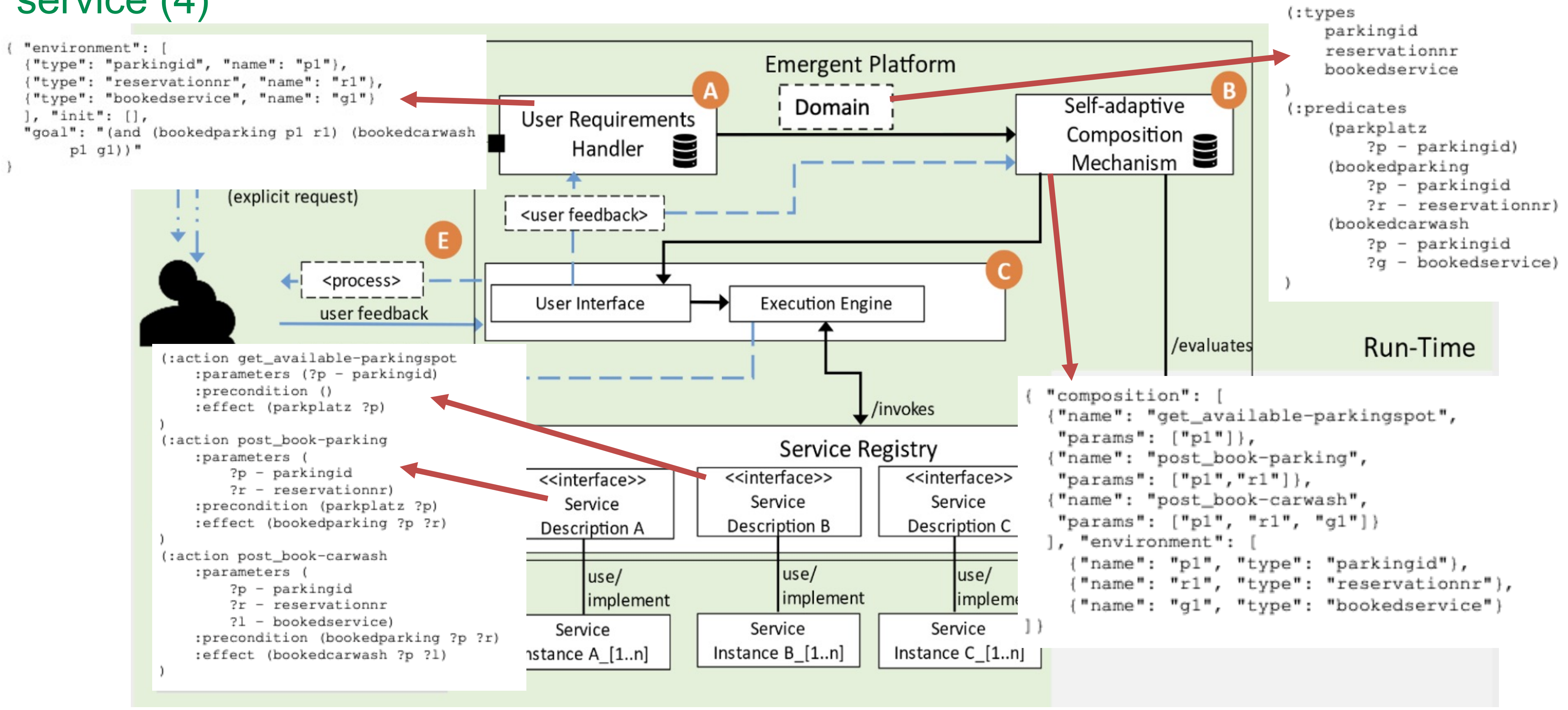
```
(:types
    parkingid
    reservationnr
    bookedservice
)
(:predicates
     (parkplatz
          ?p - parkingid)
     (bookedparking
          ?p - parkingid
          ?r - reservationnr)
     (bookedcarwash
          ?p - parkingid
          ?g - bookedservice)
)
```

```
(:action get_available-parkingspot
    :parameters (?p - parkingid)
    :precondition ()
    :effect (parkplatz ?p)
)
(:action post_book-parking
    :parameters (
        ?p - parkingid
        ?r - reservationnr)
    :precondition (parkplatz ?p)
    :effect (bookedparking ?p ?r)
)
(:action post_book-carwash
    :parameters (
        ?p - parkingid
        ?r - reservationnr
        ?l - bookedservice)
    :precondition (bookedparking ?p ?r)
    :effect (bookedcarwash ?p ?l)
)
```

TU Clausthal

Institut für Software
and Systems Engineering

Motivating example for the PDDL based description to enable composition of web service (4)



```
{ "environment": [
  {"type": "parkingid", "name": "p1"},
  {"type": "reservationnr", "name": "r1"},
  {"type": "bookedservice", "name": "g1"}
  ], "init": [],
  "goal": "(and (bookedparking p1 r1) (bookedcarwash
        p1 g1))"
}
```

```
(:types
    parkingid
    reservationnr
    bookedservice
)
(:predicates
    (parkplatz
        ?p - parkingid)
    (bookedparking
        ?p - parkingid
        ?r - reservationnr)
    (bookedcarwash
        ?p - parkingid
        ?g - bookedservice)
)
```

```
(:action get_available-parkingspot
    :parameters (?p - parkingid)
    :precondition ()
    :effect (parkplatz ?p)
)
(:action post_book-parking
    :parameters (
        ?p - parkingid
        ?r - reservationnr)
    :precondition (parkplatz ?p)
    :effect (bookedparking ?p ?r)
)
(:action post_book-carwash
    :parameters (
        ?p - parkingid
        ?r - reservationnr
        ?l - bookedservice)
    :precondition (bookedparking ?p ?r)
    :effect (bookedcarwash ?p ?l)
)
```

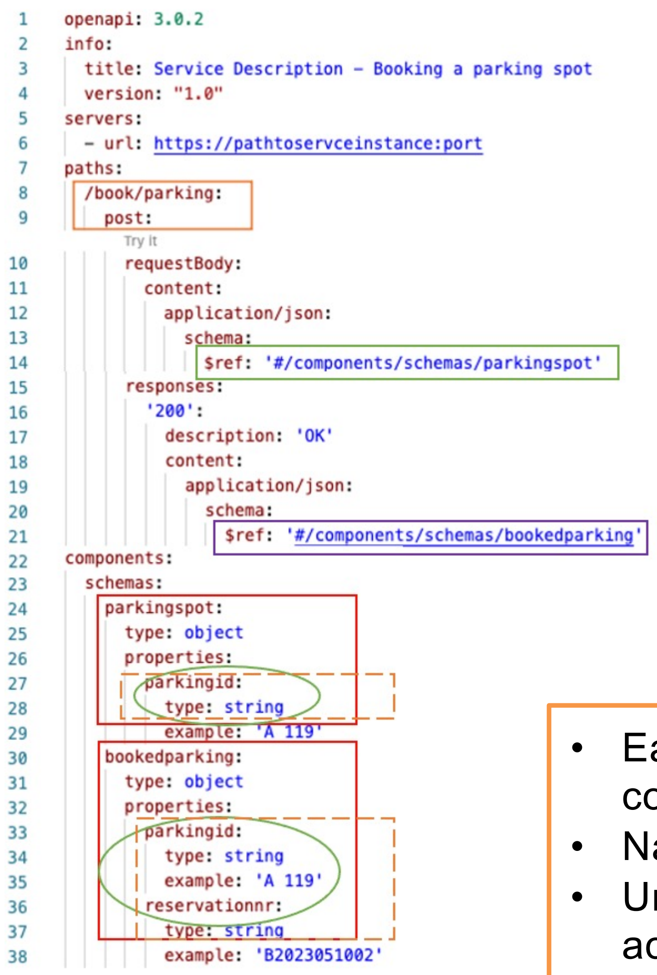
```
{ "composition": [
  {"name": "get_available-parkingspot",
   "params": ["p1"]},
  {"name": "post_book-parking",
   "params": ["p1","r1"]},
  {"name": "post_book-carwash",
   "params": ["p1", "r1", "g1"]}
  ], "environment": [
    {"name": "p1", "type": "parkingid"},
    {"name": "r1", "type": "reservationnr"},
    {"name": "g1", "type": "bookedservice"}
]}
```

Emergent Platform
A
User Requirements Handler
Domain
B
Self-adaptive Composition Mechanism
(explicit request)
<user feedback>
E
<process>
user feedback
C
User Interface
Execution Engine
/evaluates
Run-Time
/invokes
Service Registry
<<interface>> Service Description A
<<interface>> Service Description B
<<interface>> Service Description C
use/ implement
use/ implement
use/ impleme
Service Instance A_[1..n]
Service Instance B_[1..n]
Service Instance C_[1..n]

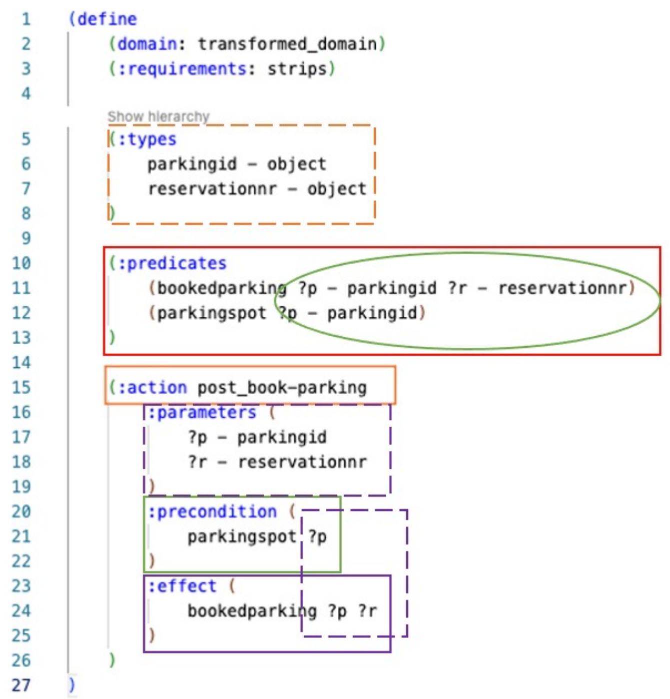
TU Clausthal

Institut für Software and Systems Engineering

Rules (1/5): Action creation



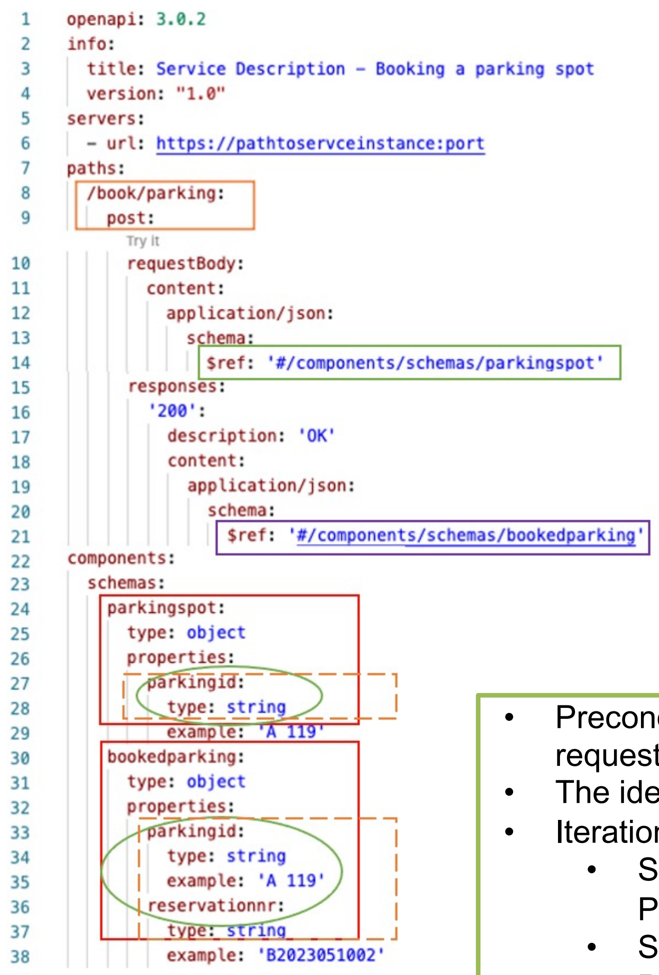
```
openapi: 3.0.2
info:
  title: Service Description - Booking a parking spot
  version: "1.0"
servers:
  - url: https://pathtoservceinstance:port
paths:
  /book/parking:
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/parkingspot'
      responses:
        '200':
          description: 'OK'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/bookedparking'
components:
  schemas:
    parkingspot:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
    bookedparking:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
        reservationnr:
          type: string
          example: 'B2023051002'
```



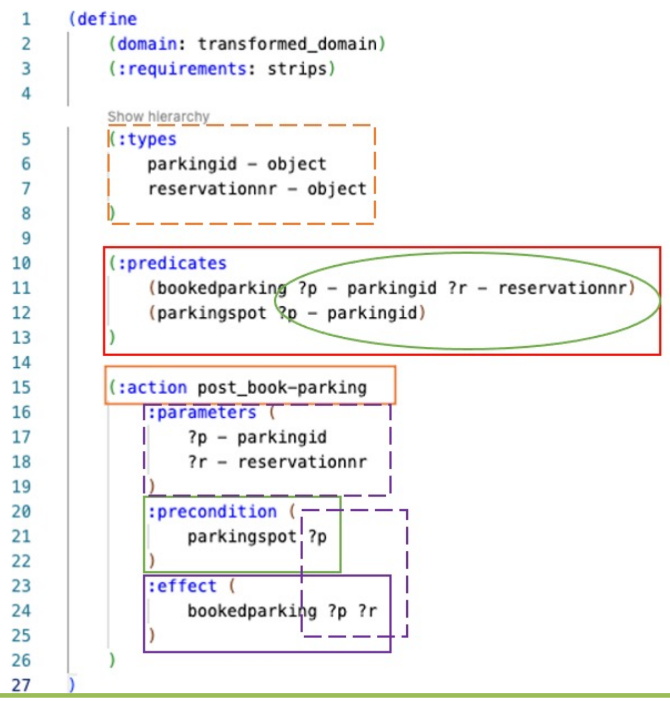
```
(define
    (domain: transformed_domain)
    (:requirements: strips)

    (:types
        parkingid - object
        reservationnr - object
    )

    (:predicates
        (bookedparking ?p - parkingid ?r - reservationnr)
        (parkingspot ?p - parkingid)
    )
    (:action post_book-parking
        :parameters (
            ?p - parkingid
            ?r - reservationnr
        )
        :precondition (
            parkingspot ?p
        )
        :effect (
            bookedparking ?p ?r
        )
    )
)
```

- Each method of each path in the OpenAPI Spec corresponds to one Action (PDDL)
- Naming schema method_path
- Unique identifiers (optional) to prevent duplicated actions across multiple OpenAPI Specs

Rules (2/5): Precondition collection



```
openapi: 3.0.2
info:
  title: Service Description - Booking a parking spot
  version: "1.0"
servers:
  - url: https://pathtoservceinstance:port
paths:
  /book/parking:
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/parkingspot'
      responses:
        '200':
          description: 'OK'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/bookedparking'
components:
  schemas:
    parkingspot:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
    bookedparking:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
        reservationnr:
          type: string
          example: 'B2023051002'
```



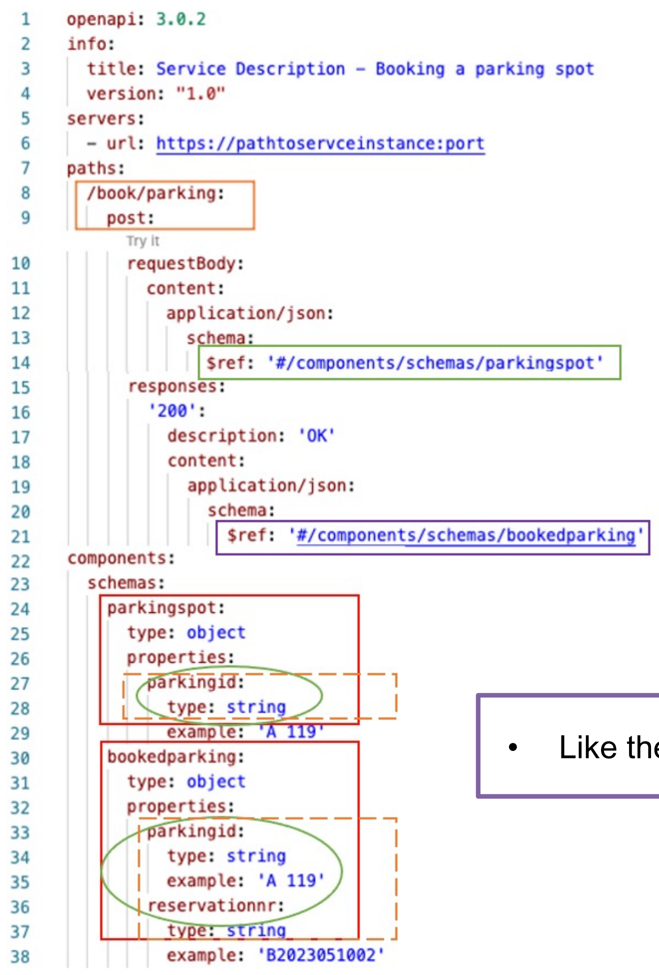
```
(define
    (domain: transformed_domain)
    (:requirements: strips)

    (:types
        parkingid - object
        reservationnr - object
    )

    (:predicates
        (bookedparking ?p - parkingid ?r - reservationnr)
        (parkingspot ?p - parkingid)
    )

    (:action post_book-parking
        :parameters (
            ?p - parkingid
            ?r - reservationnr
        )
        :precondition (
            parkingspot ?p
        )
        :effect (
            bookedparking ?p ?r
        )
    )
)
```

- Preconditions are gathered by processing the parameters, such as the requestBody
- The idea is that parameters must be present before calling the web service
- Iteration over the schema of the requestBody is performed
    - Schema elements of type "object" are added as new Precondition Predicates
    - Schema elements of primitive types become Parameters of the parent Precondition

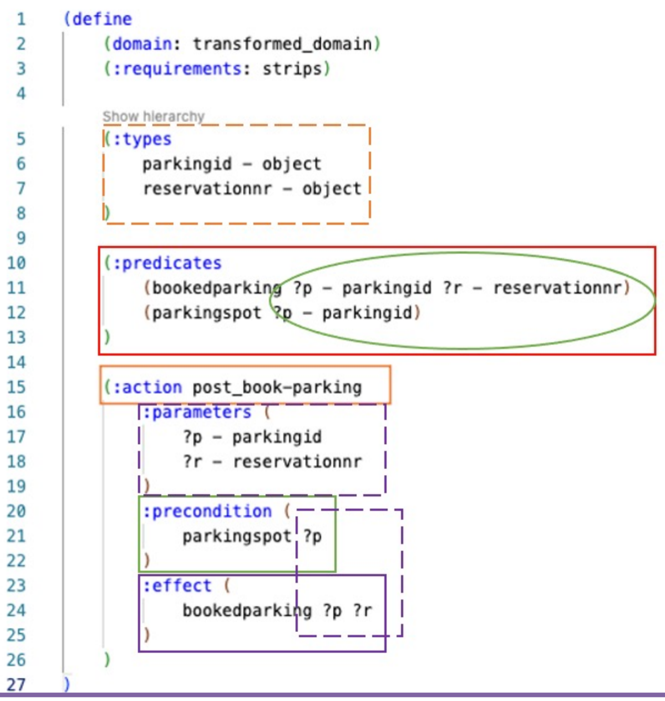
TU Clausthal

Institut für Software and Systems Engineering

Rules (3/5): Effect collection



```yaml
openapi: 3.0.2
info:
  title: Service Description - Booking a parking spot
  version: "1.0"
servers:
  - url: https://pathtoservceinstance:port
paths:
  /book/parking:
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/parkingspot'
      responses:
        '200':
          description: 'OK'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/bookedparking'
components:
  schemas:
    parkingspot:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
    bookedparking:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
        reservationnr:
          type: string
          example: 'B2023051002'
```



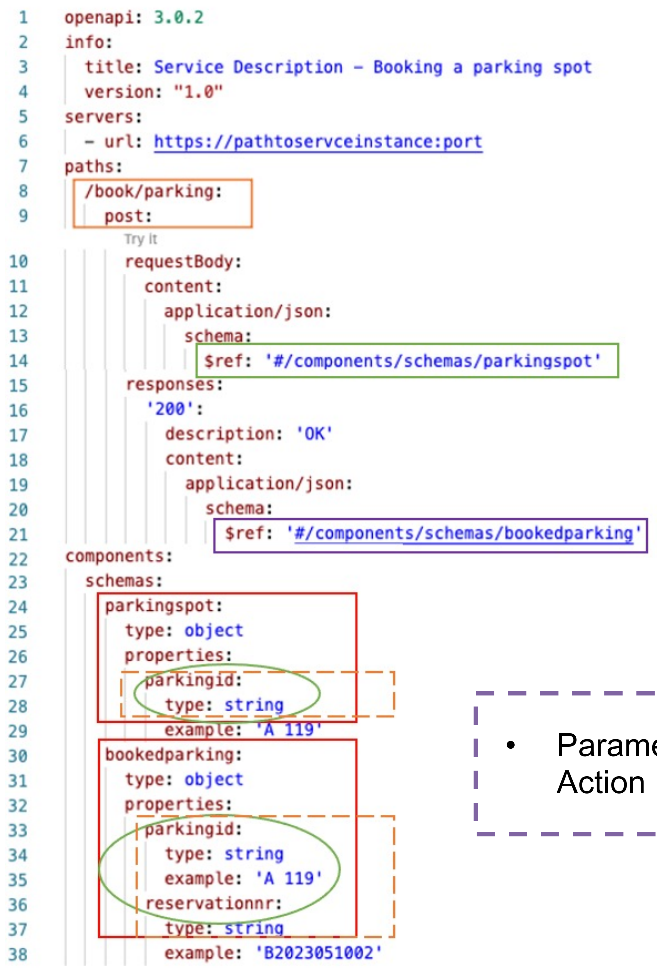
```
(define
    (domain: transformed_domain)
    (:requirements: strips)

    (:types
        parkingid - object
        reservationnr - object
    )

    (:predicates
        (bookedparking ?p - parkingid ?r - reservationnr)
        (parkingspot ?p - parkingid)
    )

    (:action post_book-parking
        :parameters (
            ?p - parkingid
            ?r - reservationnr
        )
        :precondition (
            parkingspot ?p
        )
        :effect (
            bookedparking ?p ?r
        )
    )
)
```

- Like the Precondition collection, but with information from the responseBody

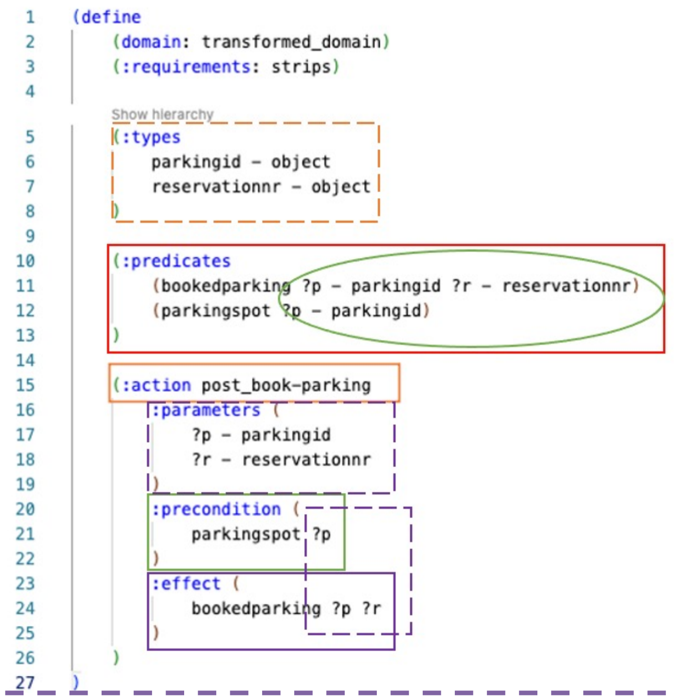
TU Clausthal

Institut für Software
and Systems Engineering

Rules (4/5): Parameter collection



```
openapi: 3.0.2
info:
  title: Service Description - Booking a parking spot
  version: "1.0"
servers:
  - url: https://pathtoservceinstance:port
paths:
  /book/parking:
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/parkingspot'
      responses:
        '200':
          description: 'OK'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/bookedparking'
components:
  schemas:
    parkingspot:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
    bookedparking:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
        reservationnr:
          type: string
          example: 'B2023051002'
```



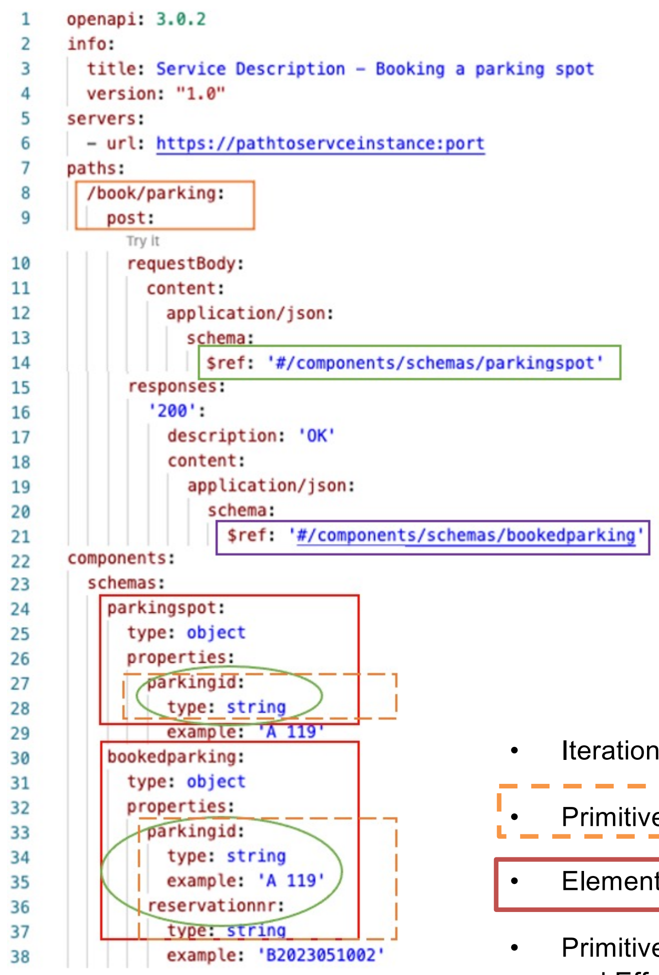
```
(define
    (domain: transformed_domain)
    (:requirements: strips)

    (:types
        parkingid - object
        reservationnr - object
    )

    (:predicates
        (bookedparking ?p - parkingid ?r - reservationnr)
        (parkingspot ?p - parkingid)
    )

    (:action post_book-parking
        :parameters (
            ?p - parkingid
            ?r - reservationnr
        )
        :precondition (
            parkingspot ?p
        )
        :effect (
            bookedparking ?p ?r
        )
    )
)
```

- Parameters from the Preconditions and Effects are collected and added as Action Parameter

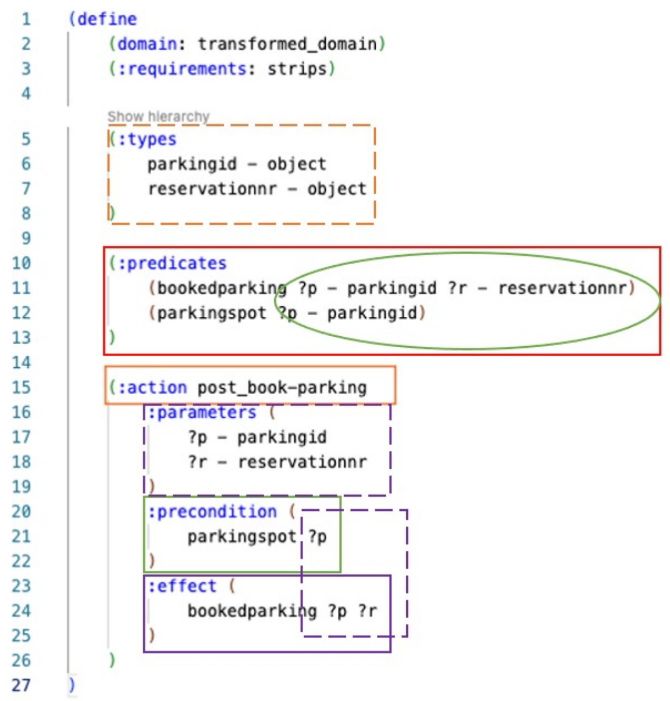
TU Clausthal

ISSE Institut für Software and Systems Engineering

Rules (5/5): Creation of the PDDL Domain



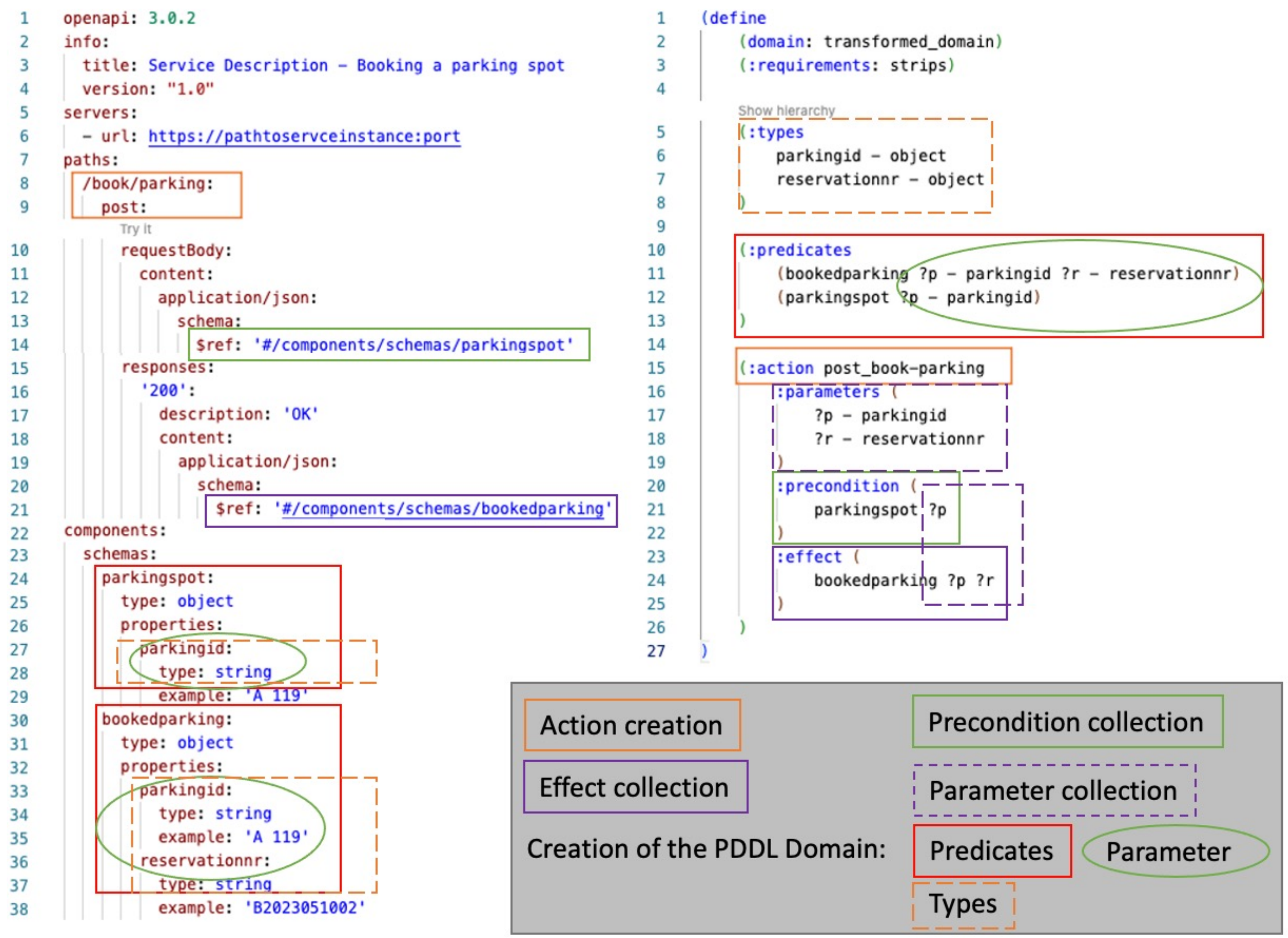
```
openapi: 3.0.2
info:
  title: Service Description - Booking a parking spot
  version: "1.0"
servers:
  - url: https://pathtoservceinstance:port
paths:
  /book/parking:
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/parkingspot'
      responses:
        '200':
          description: 'OK'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/bookedparking'
components:
  schemas:
    parkingspot:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
    bookedparking:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
        reservationnr:
          type: string
          example: 'B2023051002'
```



```
(define
  (domain: transformed_domain)
  (:requirements: strips)

  (:types
    parkingid - object
    reservationnr - object
  )

  (:predicates
    (bookedparking ?p - parkingid ?r - reservationnr)
    (parkingspot ?p - parkingid)
  )

  (:action post_book-parking
    :parameters (
      ?p - parkingid
      ?r - reservationnr
    )
    :precondition (
      parkingspot ?p
    )
    :effect (
      bookedparking ?p ?r
    )
  )
)
```

- Iteration over all relevant requestBodies and responseBodies in the OpenAPI Spec
- Primitive elements are added as types in the PDDL Domain without duplicates
- Elements of type "object" are added as predicates in the domain.
- Primitive child elements become their parameter types to conform to the Preconditions and Effects of the Actions

Mapping Example - Overview



```yaml
openapi: 3.0.2
info:
  title: Service Description - Booking a parking spot
  version: "1.0"
servers:
  - url: https://pathtoservceinstance:port
paths:
  /book/parking:
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/parkingspot'
      responses:
        '200':
          description: 'OK'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/bookedparking'
components:
  schemas:
    parkingspot:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
    bookedparking:
      type: object
      properties:
        parkingid:
          type: string
          example: 'A 119'
        reservationnr:
          type: string
          example: 'B2023051002'
```

```lisp
(define
    (domain: transformed_domain)
    (:requirements: strips)

    (:types
        parkingid - object
        reservationnr - object
    )

    (:predicates
        (bookedparking ?p - parkingid ?r - reservationnr)
        (parkingspot ?p - parkingid)
    )

    (:action post_book-parking
        :parameters (
            ?p - parkingid
            ?r - reservationnr
        )
        :precondition (
            parkingspot ?p
        )
        :effect (
            bookedparking ?p ?r
        )
    )
)
```

TU Clausthal

ISSE Institut für Software and Systems Engineering

Conclusion

- The usefulness of transforming OpenAPI into PDDL has been motivated, showcasing its practical value.
- The transformation has been aligned with our goal of developing a self-adaptive platform.
- Rules for the transformation have been defined and applied successfully.
- An example was provided to demonstrate the transformation process and its outcomes.

Future Work

- Future work includes exploring methods to ensure the quality of the transformation, leveraging expert knowledge to enhance coherence between inputs and outputs.
- Further extension of the transformation is planned to incorporate more technical details, such as
  - different response codes,
  - parameter requirements, and
  - content types in web service descriptions.

TU Clausthal

Institut für Software and Systems Engineering

BACKUP

Rules

- Action creation:
  - Each method of each path in the OpenAPI Spec corresponds to one Action (PDDL)
  - Naming schema method_path
  - Unique identifiers (optional) to prevent duplicated actions across multiple OpenAPI Specs
- Precondition collection:
  - Preconditions are gathered by processing the parameters, such as the requestBody
  - The idea is that parameters must be present before calling the web service
  - Iteration over the schema of the requestBody is performed
    - Schema elements of type "object" are added as new Precondition Predicates
    - Schema elements of primitive types become Parameters of the parent Precondition
- Effect collection:
  - Like the Precondition collection, but with information from the responseBody
- Parameter collection:
  - Parameters from the Preconditions and Effects are collected and added as Action Parameter
- Creation of the PDDL Domain:
  - Iteration over all relevant requestBodies and responseBodies in the OpenAPI Spec
  - Primitive elements are added as types in the PDDL Domain without duplicates
  - Elements of type "object" are added as predicates in the domain.
    - Primitive child elements become their parameter types to conform to the Preconditions and Effects of the Actions