Karlsruhe Institute of Technology

1st Workshop on Programming the Semantic Web

Toward Data-driven Programming for RESTful Linked Data

Steffen Stadtmüller, Dr. Andreas Harth

AIFB, Karlsruhe Institute of Technology, DE

Boston, US, MA, 11/11/2012

INSTITUTE OF APPLIED INFORMATICS AND FORMAL DESCRIPTION METHODS (AIFB)

AIFB

www.aifb.kit.edu

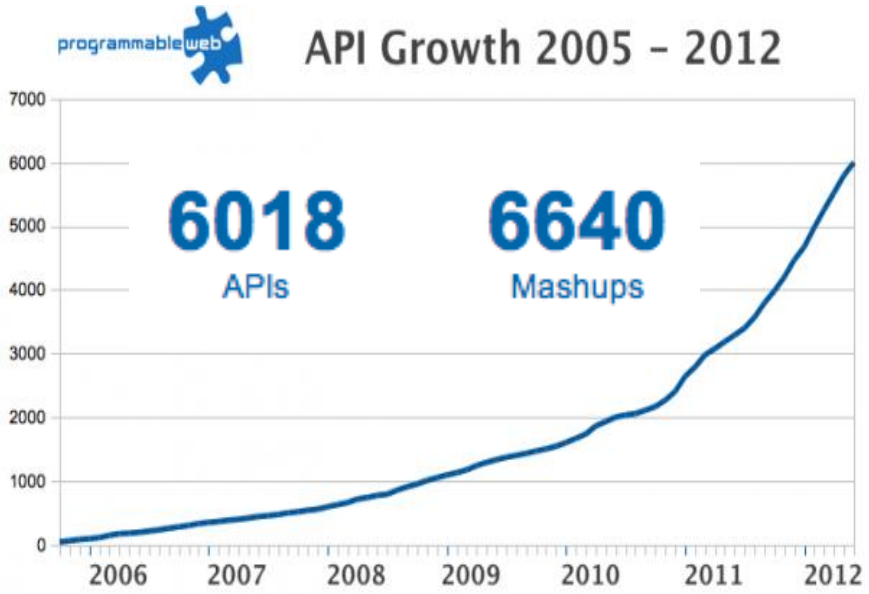
Agenda

- Motivation
  - Programming with REST
  - Challenges
- REST Execution Language
  - Formal Grounding
  - Client Behaviour Defined by Rules
- REST Service Execution Engine
  - Rete Query Engine
  - Transition System
- Conclusion
  - Related Work
  - Future Work
  - Summary

MOTIVATION

Motivation

- The Web today is not only about serving static data:
  - APIs are used to trigger functionalities in the web and the real world as well (e.g., ordering a pizza or solving a re-captcha)
  - Programmableweb.com lists ~6000 APIs[1]
- An important role plays **Representational State Transfer (REST)**
  - Software architecture for client–server interaction
  - Focused on the Web architecture



[1]http://programmableweb.com

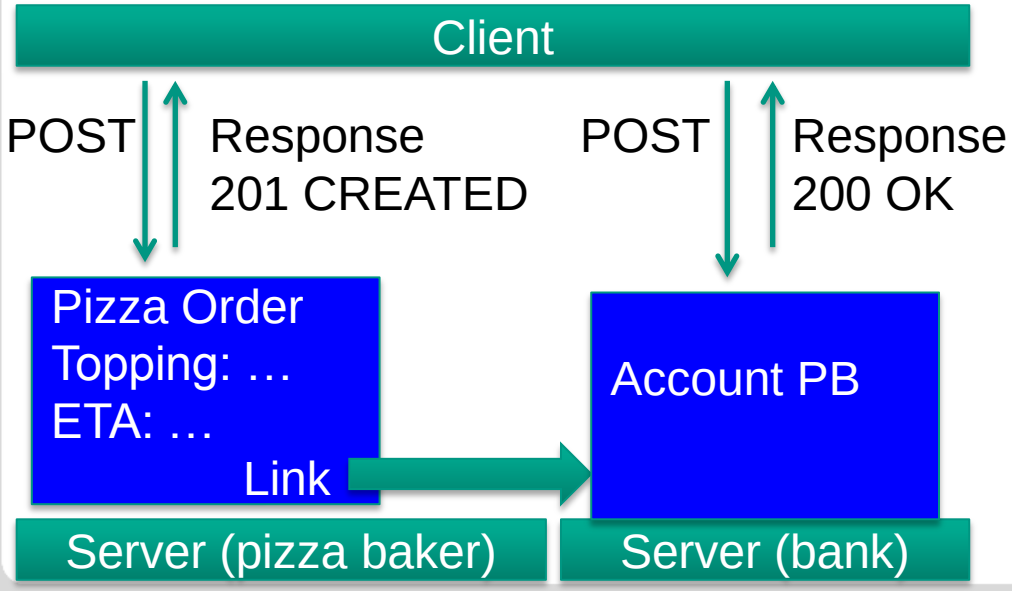
Programming with REST

- Functionality is enabled by exposing resources and allowing clients to manipulate them
  - Real world entities (e.g., car, movie, person…) are projected onto the web by making the information associated with them (their state) accessible on the Web.
  - Manipulation is possible with a constraint set of HTTP methods
  - No arbitrary function definition

HTTP Verb	Effect
GET	retrieves the representation/state of a resource identified with a URI
PUT	create or overwrite a resource identified by a client-generated URI
POST	create a resource identified by a service-generated URI
DELETE	deletes a resource (or its representation) identified with a URI

Programming with REST

- REST is focussed on loose coupling, thus enabling flexibility and robustness toward server side changes
  - Representations of resources contain links to other resources to show clients what to do next
  - Interactions should not be hard-coded
- Example:



- Client creates a new pizza order
- Pizza baker server replies with the created order including ETA and a link to his bank account
- Client can use the bank account resource to pay

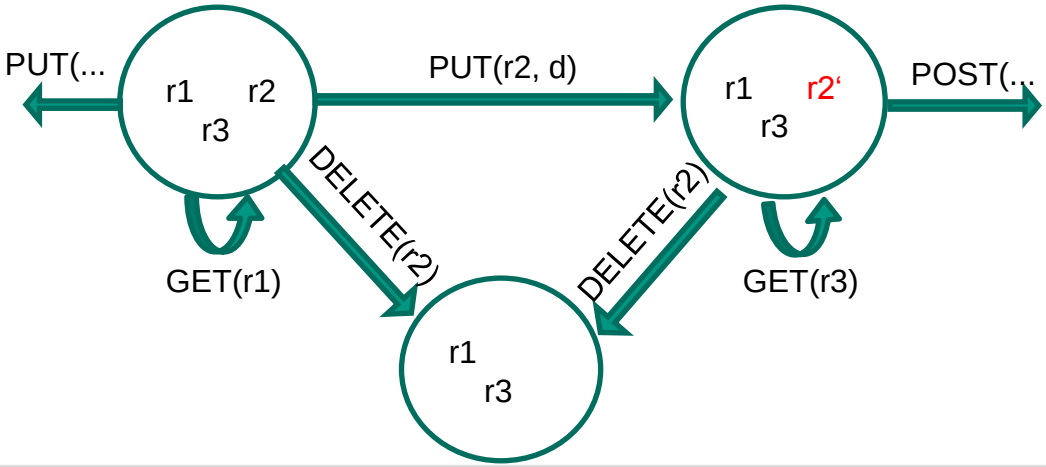
Challenges to Address

- REST allows service providers to use arbitrary formalisms to represent resources and links
  - ➡ Developers have to gain a deep understanding of every API by reading textual descriptions.
- Applications (clients) are supposed to follow links as found during runtime of the application. However, developers have to define their desired interaction at design time.
  - ➡ Developers have to write individually tailored code to consume services in applications.
- **Approach**:
  - Linked APIs: Linked Data / RDF as uniform semantic data model for resources
  - Declarative rule-based execution language for composition and orchestration
  - Execution Engine to perform the actual interaction

REST EXECUTION LANGUAGE

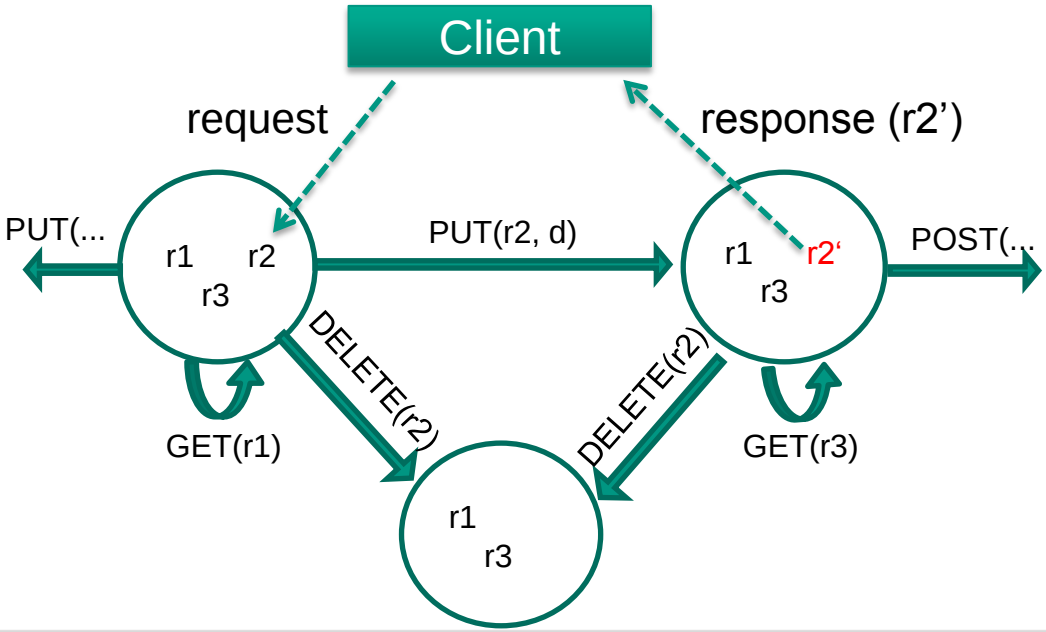
REST State Transition System

- A Linked API establishes a state transition system on the Web:
  - Every state in the transition system is the collection of all states of the resources exposed by the service
  - An HTTP method applied to a resource potentially changes the state of resources and is therefore a transition in the system.



REST State Transition System

- A Linked API establishes a state transition system on the Web:
  - Every state in the transition system is the collection of all states of the resources exposed by the service
  - To every application of an HTTP method, the server replies with a defined response, that details the effected changes



Rule Execution Language

- The state transition system builds the formal grounding for a rule execution language
- A rule is of the form

$$m(r, d) \leftarrow Q$$

  - m: HTTP method (transition) to be applied
  - r: addressed resource (in the current state)
  - d: input data (potentially empty)
  - Q: Query as condition under which the rule is to be executed

- Intuitively a rule defines what transition should be executed as subject to conditions on the current state

Query Bindings in Rule Bodies

- Since resources are represented with RDF (graph model), RDF queries (graph pattern) are used as rule bodies
- If a query delivers results, the corresponding transition is executed

- The query results (i.e. bindings), can be used to construct the input data for HTTP method, if required

$$M(r, d(?x)) \leftarrow Q_2(?x)$$

Utilization of Links

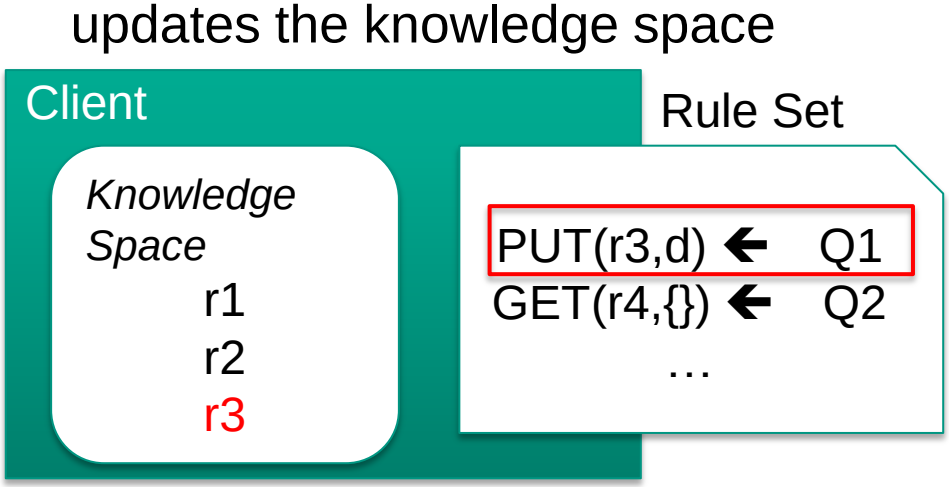
- Linked Data resources, as used by Linked APIs contain inherently links in their representation to other resources
  - Traditionally Linked Data resources can not be manipulated
  - But linked resources can also accessible for RESTful interaction
- To leverage links enables the main advantage of REST (i.e., runtime flexibility)

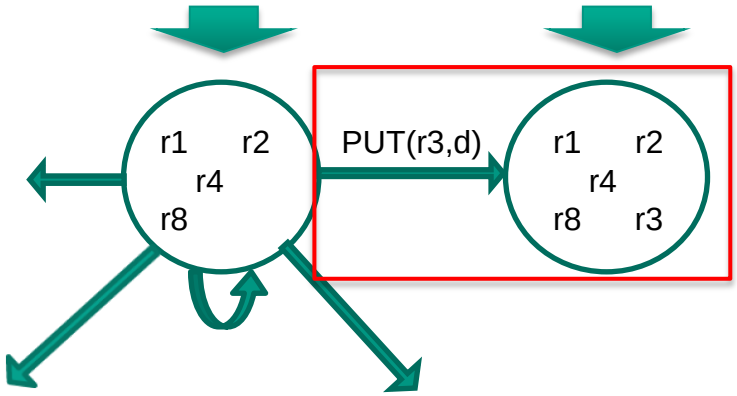
- Rule heads support the selection of resources as bindings produced by the query in the rule body:

$$M(?x, d) \leftarrow Q_2(?x)$$

Client Behaviour Defined by Rules

- Clients maintain their own knowledge space that contains information about the current state of (some) resources
  Rule bodies (queries) are constantly evaluated over the knowledge space
  - If a query matches the information in the knowledge space, the method defined in the rule head is executed
  - The server informs the client about the effected changes, which updates the knowledge space

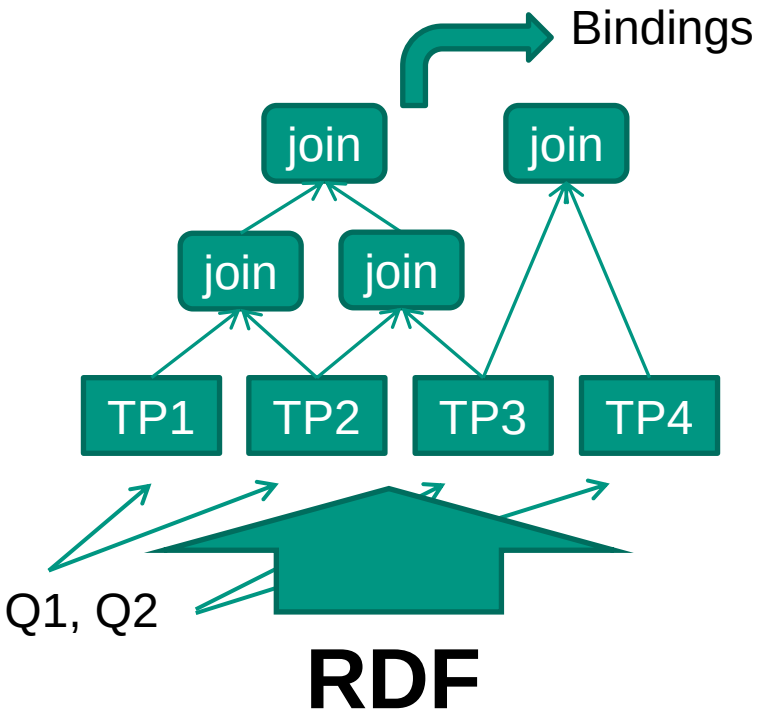




REST SERVICE EXECUTION ENGINE

Service Execution Engine - Rete

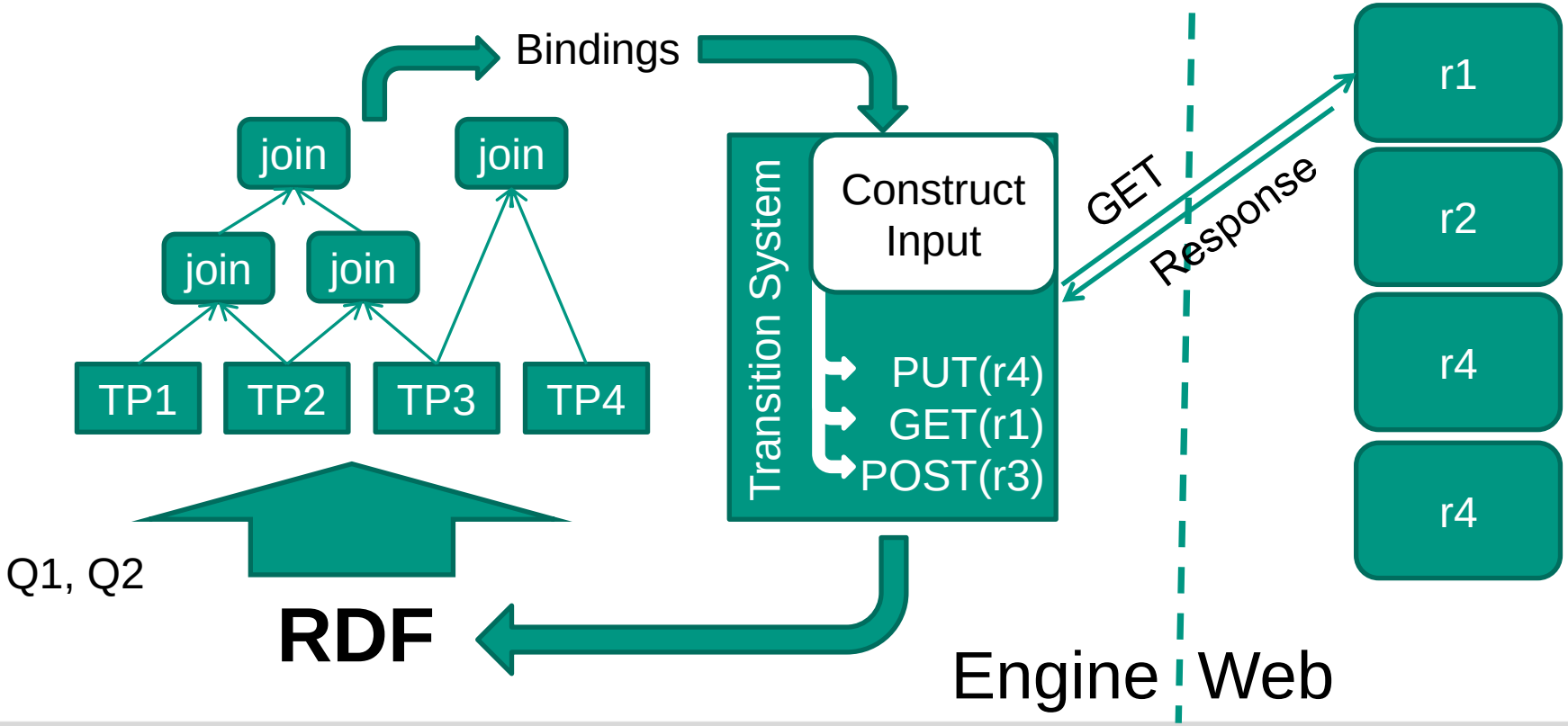
- The Execution Engine uses a multi query engine at its core, capable of processing multiple queries in parallel based an the Rete algorithm



- The queries are divided in their atomic parts (triple pattern)
- A plan (β-net) is established, how the results of the atomic queries have to be joined to receive the result of the overall queries
- RDF is sent through the net to receive the bindings of all queries

Service Execution Engine – Transition System

- If a binding for a query is found a transition system executes the HTTP method of the corresponding rule
- The RDF response is sent through the β-net



Current Status

- Engine works for a monotone subset of rules, i.e., only creation or extension of resources possible
- Deletion and modification of resources can lead to non-deterministic behaviour in a declarative rule set:

„Delete B if A exists.“

„Create C if B exists.“

Knowledge Space: {A, B}

⇨ If C is created, depends on the order of evaluation.

- **Approach**: Stratification, i.e. layering of rules, to ensure a deterministic evaluation order

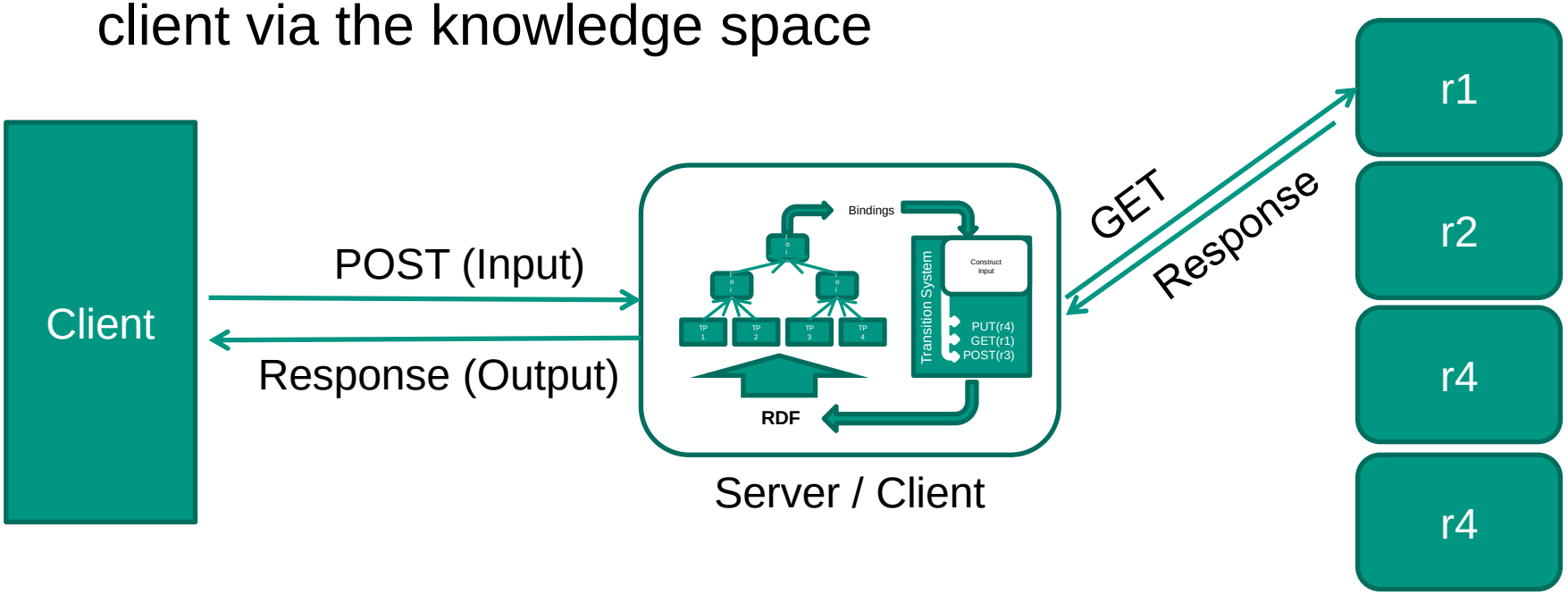
Layered Composition

- The execution language supports the notion of input and output defined with graph patterns:
  - **Input(Q):** Defines the structure of data that can initially be imported in the knowledge space to start the interaction
  - **Output(Q):** Defines the structure of data that is extracted from the knowledge space after the interaction

- A defined interaction (i.e. rule set) can be deployed on the Web as resource itself
  - Input can be HTTP POSTed to the composed resource
  - Output is returned in the response

Layered Composition

- The composed resource is server for other clients and client for resources on higher layers
- Note: state of resources on higher layers can reach the client via the knowledge space



CONCLUSION

Related Work

- Kopecky et al.: Semantic Web Services and REST: WSMO-Lite
- Pautasso: BPEL extension to compose WSDL-wrapped REST services
- Hernandez et al.: Model for REST based on a combination of pi-calculus and an extension of triple space computing (Simperl et al.)

REST is based on another kind of abstraction than traditional Service and SWS technologies: the resource. We favour a more data-driven approach

- Verborgh et al.: RESTdesc, RESTful LD resources and composition with N3-reasoning
- Bonetta et al.: script language S to develop REST resources with focus on performance due to parallel calculation

We focus on the dynamic reaction on resource states for flexible behaviour

- Krummenacher et al: Process Spaces; RESTful Linked Open Services read and write to a shared space
- Speiser et al.: Linked Data services; RESTful Linked Data

Improvement on these ideas with a well defined service model and an explicit definition of intended interactions

Future Work

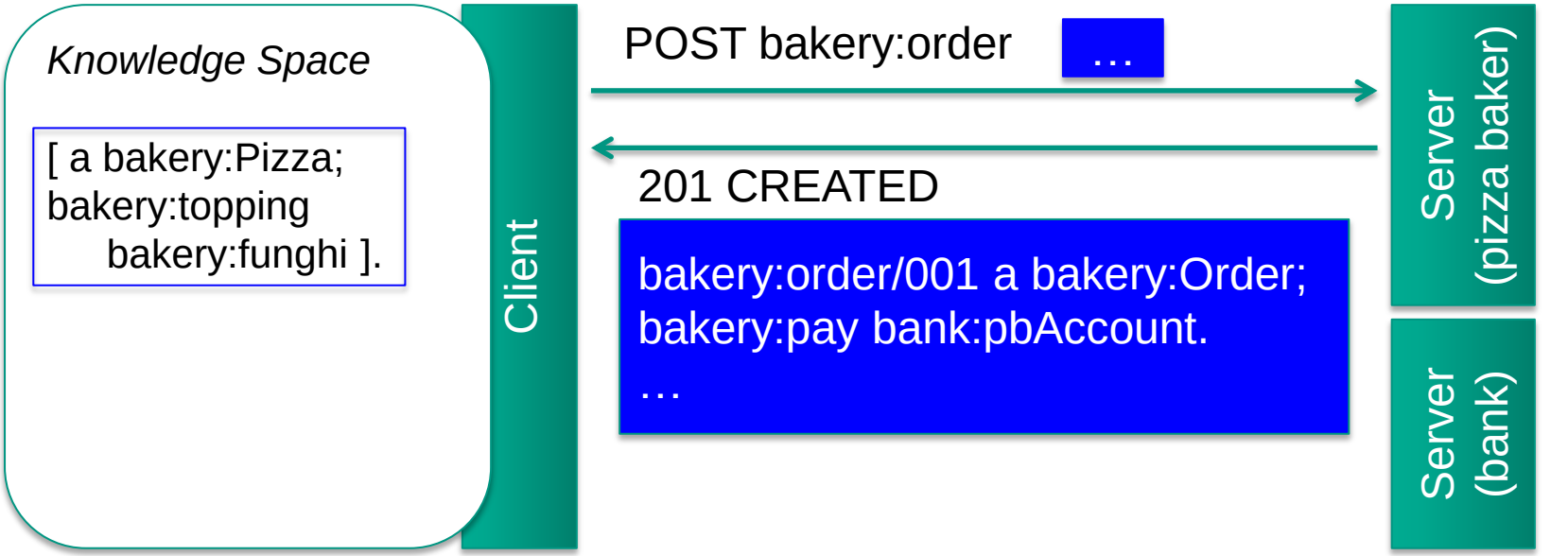
- Support for streaming resources, i.e. continuous state updates
- Ensure deterministic behaviour with non monotone rule sets
- Handling of non-Linked API resources (i.e., not Linked Data based)
  - Non-Linked Data resources (e.g., binary data, pictures, XML) can be stored separately by the engine and hooked in the knowledge space with a created URI
- Validation, verification, auto-completion with the help of service descriptions

Summary

- REST is a lightweight software architecture especially suited for client server interaction
- Programs are based on the manipulation of (Linked Data) resources on the Web
- *Goals:*
  - Mitigation of manual integration effort
  - Preservation of runtime flexibility (i.e., follow links) and capability to dynamically react on resource states
  - High scalability
- *Approach:* Declarative rule-based execution language + engine, based on resources with Linked Data as uniform semantic data model

Thank you

Example - Revisited



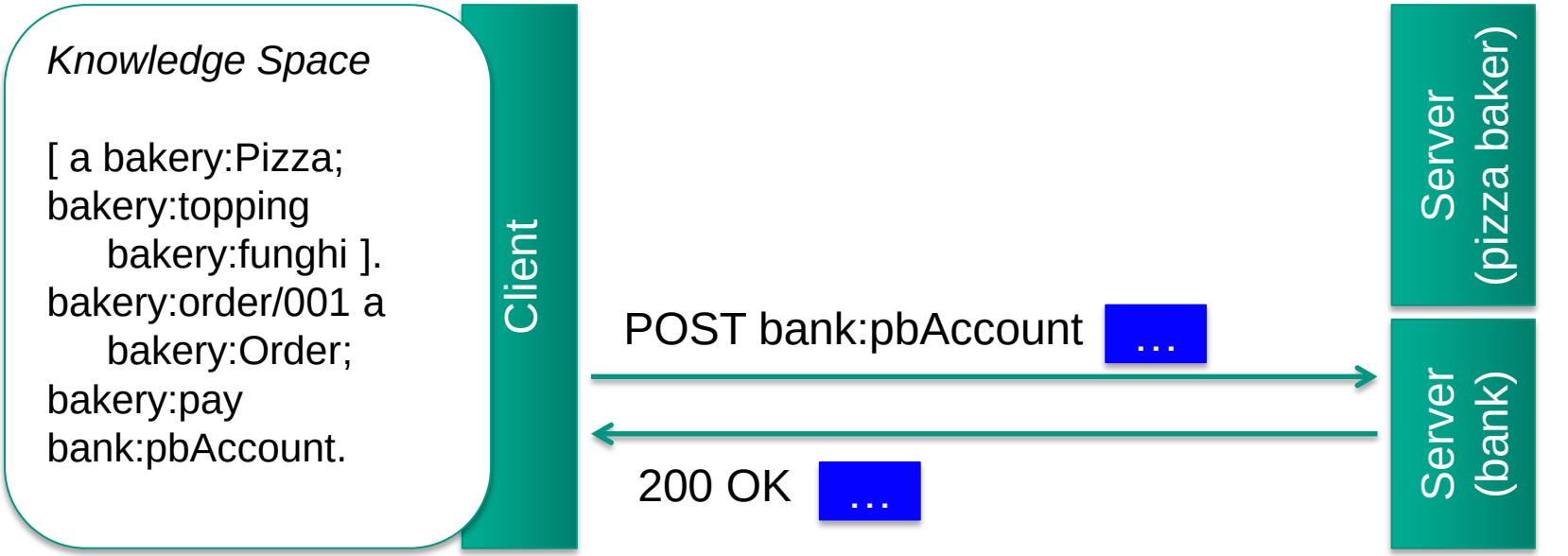
POST (bakery:order,
{ ?p a bakery:Pizza.
?p bakery:topping ?t. })

{ ?p a bakery:Pizza.
?p bakery:topping ?t. })

POST (?acc,
{ _:c a bank:Creditcard.
_:c bank:cardnumber “1234”. })

{ ?o a bakery:Order.
?o bakery:pay ?acc }

Example - Revisited



POST (bakery:order,
{ ?p a bakary:Pizza.
?p bakery:topping ?t. })

{ ?p a bakary:Pizza.
?p bakery:topping ?t. })

POST (?acc,
{ _:c a bank:Creditcard.
_:c bank:cardnumber “1234”. })

{ ?o a bakery:Order.
?o bakery:pay ?acc }

Backup: REST State Transition System – Formal Definition

Definition 1. *An RSTS is defined as a 6-tuple* $RSTS = \{R, \Sigma, I, O, M, \delta\}$ *with*

- *a set of resources* $R = \{r_1, r_2, ...\}$
- *a set of states* $\Sigma = \{\sigma_1, ..., \sigma_m\}$ *with* $\sigma_k = (\bigcup_{r_i \in R} \overline{r_i^k})$ *a complete state of the RSTS with*
  - $\overline{r_i^k}$ *the RDF representation of the state of* $r_i \in R$ *in state* σ_k
- *input alphabet* $I = \{(r, g) : R \times G\}$ *where*
  - G *the set of all possible RDF graphs*
- *output alphabet* $O = \{(c, o) : C \times 2^{\overline{R}}$ *where*
  - C *the set of all HTTP status codes*
  - $\overline{R} = \bigcup_{k=1}^{m} \bigcup_{r_i \in R} \overline{r_i^k}$, *the set of all possible states of all resources*
- *the set of HTTP methods*[5] $M = \{GET, DELETE, PUT, POST\}$
- *update function* $\delta : \Sigma \times I \to \Sigma \times O$, *which can be decomposed in*
  - *output functions* $\delta_\mu^o : \Sigma \times I \to O$ *for every* $\mu \in M$ *given by*
    - $\delta_{get}^o(\sigma_k, (r_i, \emptyset)) = (c\,,\, \overline{r_i^k})$
    - $\delta_{delete}^o(\sigma_k, (r_i, \emptyset)) = (c\,,\, \emptyset)$
    - $\delta_{put}^o(\sigma_k, (r_i, g)) = (c\,,\, \sigma_{k+1} \backslash \sigma_k)$
    - $\delta_{post}^o(\sigma_k, (r_i, g)) = (c\,,\, \sigma_{k+1} \backslash \sigma_k)$
  - *state change functions* $\delta_\mu^s : \Sigma \times I \to \Sigma$ *for every* $\mu \in M$ *given by*
    - $\delta_{get}^s(\sigma_k, (r_i, \emptyset)) = \sigma_k$
    - $\delta_{delete}^s(\sigma_k, (r_i, \emptyset)) = \sigma_k \backslash \{\overline{r_i^k}\}$
    - $\delta_{put}^s(\sigma_k, (r_i, g)) = \sigma_{k+1}$
    - $\delta_{post}^s(\sigma_k, (r_i, g)) = \sigma_{k+1}$

Query Bindings in Rule Bodies

- Since Resources are represented with RDF (Graphmodel), RDF queries (Graph pattern) are used as rule bodies
- If a query delivers results, the corresponding transition is executed
- The query results (i.e. bindings), can be used to construct the input data for HTTP method, if required

  ⇨ Inputdata is constructed out of the knowledge space over which the queries are executed

- **Scalability Challange**: The interaction of a program is defined with several rules, therefore a multitude of queries have to be constantly evaluated.