2019 IEEE International Conference on Software Architecture Companion (ICSA-C)

Reinforcement Learning for IoT Interoperability

Sebastian Kotstein
Herman Hollerith Zentrum
Reutlingen University
D-71034 Böblingen, Germany
sebastian.kotstein@reutlingen-university.de

Christian Decker
Herman Hollerith Zentrum
Reutlingen University
D-71034 Böblingen, Germany
christian.decker@reutlingen-university.de

***Abstract*—In this paper, an approach is introduced how reinforcement learning can be used to achieve interoperability between heterogeneous Internet of Things (IoT) components. More specifically, we model an HTTP REST service as a Markov Decision Process and adapt Q-Learning to the properties of REST so that an agent in the role of an HTTP REST client can learn the semantics of the service and, especially an optimal sequence of service calls to achieve an application specific goal. With our approach, we want to open up and facilitate a discussion in the community, as we see the key for achieving interoperability in IoT by the utilization of artificial intelligence techniques.**

***Keywords*-Internet of Things; Interoperability; Q-Learning; Markov Decision Process; Reinforcement Learning; REST**

I. Introduction

With the Internet of Things (IoT) an emerging trend has been created enabling novel applications and use cases involving many connected sensory devices, actors and other components. The concept of IoT is enabled by an already existing communication infrastructure, namely the Internet, and the associated technologies [1], like TCP/IP and HTTP, but also by a variety of new communication protocols and data formats resulting from the requirements of these new IoT applications. However, the IoT landscape is fragmented into many use case specific domains [2], which is also reflected in the landscape of available IoT technologies: Since these protocols and data formats have been developed under different criteria and use case specific requirements, they are not always compatible to each other [3]. The integration of heterogeneous components supporting different technologies into a uniform IoT environment so that these components are interoperable to each other, can be a time-consuming task which is associated with a significant manual development effort. Considering that large IoT environments may consist of hundreds, maybe even thousands of different components, manual adaptation of different protocols and data formats is not practical.

Hence, interoperability is one of the key challenges and a fundamental research topic in the IoT area [4][5][6]. In general, interoperability can be defined as "The ability of heterogeneous IT networks, applications, or components to exchange and use information, that is, to 'talk and understand' each other" [7]. Adapted to the context of IoT, this means that heterogeneous components can share data and functions so that a desired objective (defined through the application or use case) can be achieved, regardless of the utilized communication protocol or exchanged data format. To be more detailed and in accordance with [8], three levels of interoperability can be defined for achieving horizontal interoperability between different components, where each level covers a different facet of interoperation and communication:

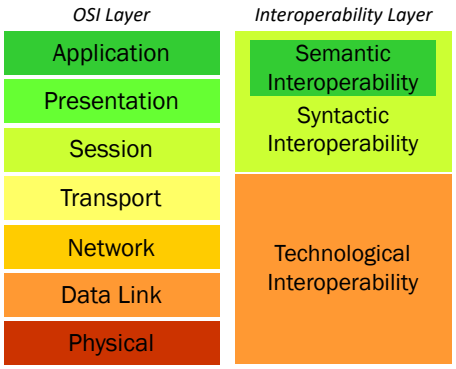

Figure 1. Mapping of interoperability levels to OSI layers

Technological Interoperability is given, if both components are physically connected to each other and can transmit any kind of information on the transmission layer [8]. This level of interoperability addresses the lower transmission layers of the OSI model (i.e. the Physical, Data Link, Network and Transport layer) and the corresponding protocols.

Syntactic Interoperability is provided, if the data format, including encodings, as well as the communication interface format is clearly defined and agreed between both components [8]. The abstract term "communication interface format" refers to the protocol which is used on the Application layer and that is offered as communication

978-1-7281-1876-5/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSA-C.2019.00010

interface by the respective component. Similar to technological interoperability, it is not necessarily required that both components agree on the same protocol as long as there is a possibility for adaptation of two different protocols (same applies for the format and encoding of the exchanged data embedded in the protocol(s)).

Semantic Interoperability is given, if both components agree on the same information model describing the meaning of exchanged information as well as of the provided communication interfaces [8] (i.e. the meaning of exchanged messages with respect to the used protocol).

Figure 1 illustrates the mapping of these three interoperability levels defined in [8] to the OSI layers. Note that the mapping is not strict and the boundaries between the OSI layers might vary depending on the implementation of the respective application. The approaches and solutions for solving interoperability issues on all levels can be mostly classified into two categories: On the one hand, standardization is considered as a driver for interoperability, especially by the industry [4]. On the other hand, adapters and gateways are developed and used [4]. Furthermore, and in accordance with standardization, open standards and formats are promoted by the community [6] as they are commonly well documented and therefore simplifying the integration of IoT components. Similarly, the approach to agree not only on open standards but also on one universal IoT standard, is popular in research and industry, but controversial as well (such an attempt has been started with ZigBee 3.0, for instance). We are convinced, that the IoT market will not agree on one universal standard as the IoT landscape is too versatile and fragmented. The problem of the diversity of protocols and standards is likely to become even more severe.

A promising approach, however, is the Web of Things, as it relies on established web technologies, that already offer a high degree of interoperability and flexibility. More specifically, the Web of Things follows the idea that physical Things are handled like any other web resource and can be queried and manipulated over well-established protocols namely HTTP on the application layer and Internet protocols on the transmission layer [9]. As the Web of Things allows the interconnection between different IoT components and platforms through HTTP and the associated Internet protocols, we consider syntactic as well as technological interoperability as given. Nevertheless, in terms of seamless interoperability, the Web of Things is limited, as it provides neither self-description mechanisms for communication interfaces and data formats natively, nor are these interfaces machine-interpretable. However, and similar as in the domain of Service-oriented Computing, there exists various approaches for enriching these web components with meta information providing a semantic model (i.e. a description of the meaning of interfaces and formats) such as ontologies, semantic frameworks and other meta models [10][11]. In addition, many of these approaches are supplemented by mechanisms, such that an agent-based system can read and interpret given semantic descriptions and adapt two or more components autonomously (an overview of approaches can be found in [10]). Nevertheless, it is still a high manual effort to enrich components with meta information such that they are described in a machine-interpretable way. And similar to the protocol landscape of IoT, there exists numerous proposals for semantic models and ontologies specialized on different IoT domains leading to incompatibility on the semantic layer [11]. Instead of using given meta information on the semantic level for adaptation, we suggest that an agent-based system learns the semantics of a component by trying out actions and observing the corresponding outcome:

In this paper, we present an approach how to achieve interoperability using a technique called reinforcement learning. In our approach, an HTTP REST service, as it is used by a Web of Things component, is modelled as a Markov Decision Process (MDP) so that an agent (in the role of an HTTP REST client) is able to learn how to use this service by applying a reinforcement learning method called Q-Learning, which is adapted to the properties of HTTP REST (in the following, we will use the term REST instead of HTTP REST). Such a service could be a REST API gateway controlling the lights in a smart home environment, for instance. More specifically, the agent learns a sequence of service calls to achieve an application specific goal (e.g. the sequence of calls which are required to switch on a light bulb controlled by the API gateway). As we choose HTTP/REST on the syntactic and the Internet as transmission medium on the technological interoperability level, we consider these two levels of interoperability as already given. But on the semantic level, the agent has no knowledge about the runtime behavior of the service and does not know what consequences a specific service call might have. However, the agent has the capabilities to make observations by trying out actions and can build up a semantic understanding describing the experienced behavior of the service. This semantic understanding is not comparable to the understanding a human developer would build up during the exploration of the service or by reading the service documentation, as the agent lacks of imagination nor it knows the meaning of any kind of symbol (e.g. the meaning of a specific function name). Nevertheless, we believe that even this limited semantic knowledge is sufficient to handle the service in a proper way, and therefore for achieving interoperability. Although this approach is primarily situated in the area of Web of Things due to HTTP/REST, we assume that the underlying

techniques can be adapted to other protocols, and even to other levels of interoperability. Much more, the approach is intended to give an impetus to the community and to open up a discussion of how to utilize and adapt AI techniques for solving interoperability issues on all levels.

The remaining paper is organized as follows: Section II presents the related work. Section III introduces the key concept of an MDP and gives insight into the details of modelling a REST service as an MDP. Section IV gives a brief overview about Q-Learning and adapts this learning algorithm to the properties of REST. In section V, we discuss the observations made during the modelling and adaptation and close with an outlook on the future planed research. Finally, a conclusion is given in section VI.

II. Related Work

Sequential decision problems such as MDPs as well as techniques for reinforcement learning (e.g. Q-Learning) are part of the planning domain in AI. In general, this domain encompasses techniques for finding a sequence of actions such that a specific goal can be achieved [12]. In course of our literature review, we focus on similar approaches for interoperability relying on the discipline of AI planning: According to [10] and [13], AI planning is an elementary discipline for dynamic web service composition, covering various different approaches and sub techniques of this field for composing and adapting web services as well as REST services. Examples for planning-based techniques can be found in [14] and [15]. [16] introduces an approach which is similar to our approach in terms of the interoperability levels and the used communication technologies as it exploits several REST principles like the linking of web resources forming one or multiple linked services and the basic semantics of HTTP methods such that a generic client can explore and interact with these linked services dynamically. But in comparison to our approach, these services and their resources are enriched with semantic descriptions. Instead of learning a semantic model, the client just interprets these descriptions and makes a decision on the fly. A similar approach is presented in [17]. On the other hand, there are approaches using same AI techniques as we do, but focusing on different adaptation constraints in terms of interoperability: In [18], web service composition is modeled as an MDP so that an agent can create an optimal service workflow composed of a given set of web services by applying Q-Learning. In a series of further publications, this approach is constantly improved and especially adapted to multi-agent scenarios (e.g. in [19] and [20]). [21] follows a similar approach. However, these approaches are focused on the optimal composition of a set of web services in terms of given Quality of Service attributes, instead of learning the semantics of a service. Furthermore, technological aspects of the services such as REST constraints are not focused in course of the used mechanisms. As a conclusion, the analyzed AI planning techniques either rely on a given semantic description for decision making or uses reinforcement learning in terms of Quality of Service attributes. Moreover, [4] presents a survey highlighting issues and challenges of interoperability in the context of IoT. The paper analyzes more than 30 approaches for interoperability in the form of IoT frameworks, platforms and projects. Although many of these approaches rely on adaptation and gateway applications, we cannot find any approach using AI techniques such as reinforcement learning. Due to our state-of-the-art findings, we see an urgent need to introduce a new AI-based approach for solving interoperability issues in the context of IoT and to open up a new direction.

III. Modelling a REST Service as an MDP

In this section, we introduce an approach how a REST service can be modelled as an MDP, so that an agent can autonomously learn the semantics of the service and find a policy describing the optimal use of this service to achieve an application specific goal. First of all, we present the key concepts of an MDP. Afterwards, we move into details of modelling. The techniques for learning are explained in section IV.

An MDP [12] consists of a set of states S (with an initial state s_o), a set of possible actions $A(s)$ for each state s and a transition model $P(s'|a,s)$ describing the probability of a transition from state s to state s' if action a is executed. Furthermore, there is a given reward function $R(s)$ returning a value (the reward) for reaching state s. At each time step, the process is in a certain state and the agent (in the role of a decision maker) may choose an action a in the set of available actions $A(s)$ for this state s. By applying the chosen action, the process moves into the next state s' with a certain probability described by $P(s'|a,s)$. The new state is reported to the agent. In addition, the agent receives a reward for reaching the new state s' based on the reward function (i.e. $R(s')$). The decision (i.e. the chosen action), an agent should make in a certain state, is expressed by a policy $\pi(s) \in A(s)$. Due to the stochastic nature of the environment (described by the transition model), the resulting state might look different every time a given policy is executed. Nevertheless, the quality of a policy can be measured by the expected utility $E[U]$, where the utility u^π is the sum of all rewards received by executing the policy π. The policy yielding the highest expected utility is considered as an optimal policy π^*. Since π^* describes the optimal sequence of decisions in an MDP, it is the goal of an agent to learn such an optimal policy π^* through reinforcement learning. In a real-world scenario, the agent should not only achieve the best utility by executing the optimal policy, but should also move the process into a desired target state such that the

goal is achieved. However, the concept of target states is not defined in an MDP and the agent might live forever. Therefore, the reward function must be chosen advisedly such that the optimal policy is not only yielding the highest expected utility but also steering the agent to the desired target state.

Based on this key concept, four questions arise for modelling a REST service as an MDP: (I) Which properties of a REST service can be used to express the current state of the service? (II) How can an action be defined in terms of REST such that the current state of the service may be changed? (III) How can the set of available actions for a specific state be defined for REST? (IV) How does the transition model look like? And finally, (V) how does the reward function look like? With these issues in mind, we would model a REST service as an MDP as follows:

A. Definition of State

Statelessness is one of the biggest advantages of REST, since it reduces the coupling between the service and the clients and increases scalability. But statelessness does not mean, that the service does not maintain any kind of state at all. It refers only to the application state, i.e. states which are related to current client sessions [22]. Global states, which are valid for any client and the service itself, i.e. for the whole world, are handled in REST in form of resource states. A resource state can be defined as the state of a resource at a certain point of time. Therefore, the service state s_t at time step t is the set of states of all resources handled by the service at time step $t : s_t = \{R_t\} = \{r_{t,0}, r_{t,1}, .., r_{t,n}\}$. In section IV, we describe an approach for traversing resources states.

B. Definition of Action

The state of the service changes if at least one resource is changed, created or deleted. HTTP defines a set of operators (so-called HTTP methods) for creating (POST), deleting (DELETE), modifying (PUT) or just retrieving (GET) a web resource. While DELETE and PUT cause a state transition, GET is defined as a safe method and can be used to retrieve the current state of the resource. POST is commonly used for creating a resource and causes a state transition as well, but in comparison to DELETE, PUT and GET its semantics are not further specified by REST. The resource being subject of the operation is addressed through its Uniform Resource Locator (URL). Hence, an action can be defined as the composition of an HTTP method and a URL.

C. Definition of Set of Available Actions for a State

The set of available actions for a specific state $A(s)$ is the composition of the sets of available actions for a specific resource: $A(s) = A(R) = A(r_0) + A(r_1) + ... + A(r_n)$. The service might limit the set of allowed HTTP methods for security or functionality reasons. These limitations can be individual on single resources, but also dependent on the requesting client. A mechanism for querying the list of allowed HTTP methods for a specific resource is presented in section IV.

D. The Transition Model

The transition model is defined by the service implementation. In a deterministic implementation, transition probabilities are set to 1, if the transition from state s to state s' by executing action a is valid. All other transitions probabilities are set to 0 (i.e. for those cases, the service does not implement an action for a transition from state s to s'). Invalid actions, i.e. actions which are not allowed in the current state since they are not permitted for a specific resource, not implemented due to reasons already mentioned or invalid requests (e.g. bad requests resulting into a 4XX Client Error) are not considered as valid actions in terms of an MDP. Hence, the state of the service is kept unaltered for those cases.

E. The Reward Function

It is the goal of the agent not to learn *any* policy describing *any* sequence of service calls to achieve a certain goal, but an *optimal* policy so that an *optimal* sequence of service calls can be executed for achieving this certain goal. But what does optimality in terms of a REST service means? In general, the reward function must be designed in such a way that the agent is steered to the desired target state representing the goal. However, this goal is strongly use case specific. As a consequence, the definition of the reward function is use case dependent as well: With each state transition, the reward function has to evaluate to what extent the new state contributes to the desired goal or whether this new state already represents the desired goal. At the same time, the reward function defines what optimality means as the function might reward some states higher than other states or even punishes states (i.e. gives a negative reward) which should be avoided. A very generic approach would be for instance, that the reward function punishes every transition to a state which does not represent the desired target state with a reward value of -1 and rewards only the desired target state with a positive value of 10. Of course, this approach also punishes intermediate states which must be inevitably reached for achieving the desired target state. But only through this way, the agent learns whether it is worth taking this intermediate state and the associated service call into account and accepting a negative reward for the moment, as it nevertheless leads to an improvement of the overall utility, or whether such a state should be avoided generally, as it does not lead to an improvement of the overall utility and does not contribute to the desired goal. The resulting optimal policy describes an optimal sequence of service calls in terms of *as few service calls as*

Initialize all $Q(s, a)$ with 0;
for each episode **do**
  $s \leftarrow s_0$;
  while $s \notin s_{target}$ **do**
    $a \leftarrow chooseAction(A(s), \varepsilon)$;
    $s', r \leftarrow ExecuteAction(a)$;
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_a Q(s', a') - Q(s,a)\right]$;
    $s \leftarrow s'$;
  end while
end for

Figure 2. Q-Learning algorithm

necessary. In addition, further application specific rules can be incorporated into the reward function. A reward function might punish the removal of resources with an additional negative reward if a service call with HTTP DELETE is used, for instance. For that case, the reward function must take not only the new state but also the action as well as the old state into account which is, in general, feasible for an MDP (i.e. $R(s, a, s')$) [12]. Section IV introduces an approach, how a reward function can be implemented in terms of a REST client-server architecture.

IV. Client Implementation with Q-Learning

In section III, we presented an MDP model for a REST service. Based on this model, someone could calculate easily an optimal policy, describing the optimal sequence of service calls for achieving an application specific goal, by applying the policy iteration algorithm [23], but only if the reward function as well as the transition model is known. For our MDP, we assume that the agent has not sufficient knowledge about semantics and behavior of the service which means that the transition model is unknown. Furthermore, the agent has no knowledge about the reward function and cannot estimate the reward it will receive for a state transition. Due to these limitations, we introduce a reinforcement learning algorithm called Q-Learning which allows to learn an optimal policy at runtime without knowing the transition model nor the reward function. Before we start to adapt this algorithm for our REST service model, we will give a brief overview about the basics of Q-Learning.

A. Q-Learning

Q-Learning [24] relies on the idea, that an agent learns a so-called action-utility function $Q(s, a)$ specifying the value (i.e. the utility) which can be expected in a state s by choosing and executing an action a. Based on the feedback the agent receives during active learning in the form of a reward and a new state by executing a chosen action, $Q(s, a)$ can be continuously updated and improved. The full Q-Learning algorithm is illustrated in figure 2 and works as follows [24]:

In the beginning, all $Q(s, a)$ values are set to 0. The learning process is organized in episodes (the number of episodes can be selected manually). With each episode, the agent starts from the same initial state s_0. In the first step, an action a provided by a so-called greedy function is loaded. Based on a greedy parameter ϵ this function decides whether to choose the current optimal action $a = argmax_{a \in A(s)} Q(s, a)$ with a probability of $1 - \epsilon$, or to choose a random action $a \in A(s)$ (probability of ϵ). This function guaranties a trade-off between exploration and exploitation. The greedy parameter can be adjusted with each episode such that explorations are more likely in early episodes while in late episodes the already learned knowledge is more often exploited. After executing the chosen action and receiving the reward r as well as the new state s', the agent updates $Q(s, a)$ by using the following equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$$

The existing Q-Value is partially replaced by the received reward r and the highest known Q-Value of the new state s' (i.e. $max_{a'} Q(s', a')$), but only with a certain portion which is defined by a so-called learning ratio $\alpha \in [0, 1]$. $\gamma \in [0, 1]$ is a discount factor describing the preference of the agent for the current reward over future rewards [12]. After updating $Q(s, a)$, the new state s' is set as current state s (i.e. the agent moves into this new state s') and a new iteration starts by choosing a new action $a \in A(s)$ (i.e. heading back to the first step). The episode ends, if the new state s' is an initially defined target state. For the next learning episode, the already learned Q-Values are reused such that the Q-Values are continuously improved.

B. Adaptation of Q-Learning

After explaining the basics of Q-Learning, we adapt the algorithm to the properties of REST so that an agent (in the role of the REST client) can learn the semantics of a REST service and especially how to use it. The following issues arise in the presented Q-Learning algorithm and must be adapted, before the algorithm can be applied:

After executing the chosen action, the agent should be informed about the new state s'. As a REST service does not implement any mechanism for notifying a client about its current state, the agent has to take the initiative and query the state on its own. In section III, we have defined the state s_t at time step t as the set of all resource states (i.e. $s_t = \{R_t\} = \{r_{t,0}, r_{t,1}, .., r_{t,n}\}$). Generally, resources can be queried using the HTTP GET method, but only if the URL is known for each resource. We assume that the agent has no knowledge about the resource structure of the service nor its URLs. However, as a REST service should be implemented hyper-driven [22] (This is part of the REST constraint "Hypermedia as the Engine of Application State"

(HATEOAS) and means, that resources points to other resources over hyperlinks (URLs), if there is a semantic relation between the resources), it should be possible to traverse the whole resource structure starting with an entry point (URL). Considering that this entry point is given and all resource, forming the state of the service, are linked to each other, an agent can query the state at any point of time. Furthermore, the HTTP GET method, which is used for querying, is safe and does not cause any state transition. The same solution applies for loading the initial state s_0 at the beginning of the algorithm.

Moreover, the greedy function requires the set of allowed actions $A(s)$ before an action a can be chosen. For this purpose, the advantages of a hyper-driven design can be exploited again: We have defined the set of allowed actions $A(s)$ in a specific state s as the set of allowed actions of all resources (i.e. $A(s) = A(R)$, see section III). Hyperlinks in context of HATEOAS are not only used to reference related resources such that they can be queried using the GET method, but also for documenting actions on resources. Therefore, hyperlinks can be considered as a potential indicator for a state transition. Although hyperlinks may have a semantic description (see [25]), we assume that the agent is not able to deduce the corresponding HTTP method from this description, unless it is explicitly given as a separate attribute (e.g "verb=POST"). Instead, we would recommend to use the HTTP OPTION method to ask for the list of allowed HTTP methods for a specific resource. Alternatively, the agent can naïvely assume that all methods (GET, POST, POST and DELETE) are allowed for a specific resource r and add them to the set of allowed actions $A(r)$. If the agent executes a restricted HTTP method later, it would receive an HTTP status code 405 Method not allowed [26] which is considered as an invalid action per definition and has no effect on the behavior of the MDP (see section III).

Furthermore, the agent should receive a reward after executing the chosen action. As we want to use any kind of existing REST service, we cannot assume, that the service is able to return a reward embedded into the HTTP response. Hence, we need a third component observing the activities in our world (see figure 3): If an action has been executed (1-3), the observer is notified by the agent (4) and queries the new state s' (5,6). In (7) the observer analyses this new state s', more specifically the queried resource states, and calculates a reward. Finally, the reward is forwarded to the agent (8). For the case that not only the new state s' but also the old state s and the executed action a are relevant for the calculation of the reward (i.e. $R(s, a, s')$, see section III), the agent has to inform the observer about the executed action a in (4) and the observer has to store the new state s' queried in (5,6) as old state s for the next

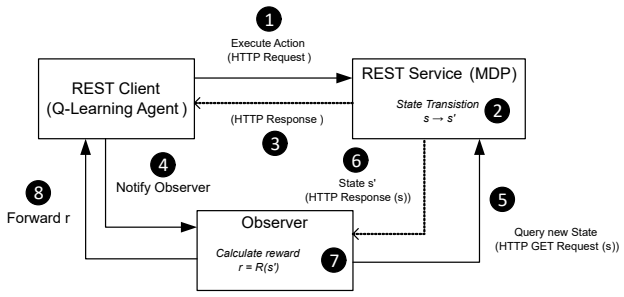

Figure 3. Observer Component

iteration. Additionally, with the beginning of every episode, the observer has to query the initial state s_0 such that it is stored as old state s initially. Alternatively, the observer could also be implemented as a proxy between the agent and the service such that it can analyze the communication between both parties and extracts the parameters which are required for calculating the reward.

V. Discussion & Outlook

The previous sections have shown, that it is possible to model a REST service as an MDP and adapt a Q-Learning algorithm so that an agent can learn the semantics of the service. The properties of REST, especially the REST constraint HATEOAS, eases the adaptation as it urges the service to be implemented hyper-driven and as a state machine. Nevertheless, during modelling and adaptation we have identified some critical points impeding a practical application. These points are discussed in the following. At the same time, these identified challenges provide an outlook on our ongoing planed research work in this area:

We assume, that querying all resource states as it is proposed in the adapted Q-Learning algorithm is not practical, if the algorithm is applied to a REST service handling thousands of resources simultaneously. In such a scenario, the agent has to send a GET request for each resource to query the full state which is not feasible, considering that this must be done after each execution. Moreover, this quantity of resources may result in an explosion of possible states which have to be distinguished. Same applies for the set of available actions in a specific state, since actions are linked to resources and with each additional resource the number of possible actions might be increased. However, it might be possible (but this depends on the implementation of the service), that an action has only an impact on a single resource instead of the whole state. Therefore, the Q-Learning algorithm should be adapted in such a way that state invariants can be detected. For the further development of our approach, we will take this into account.

Furthermore, it is currently unclear how to deal with optional or even mandatory additional parameters such as URL-Queries, additional headers or payloads which can be part of an HTTP request. It remains open, how to distinguish these combinations without running into an explosion of different actions, on the one hand. On the other hand, the agent must be aware of the syntax and semantics of the additional fields. This knowledge is currently not given. Hence, a mechanism is needed for either learning this syntax and semantics or for receiving an appropriate description.

Additionally, the Q-Learning process is organized in episodes always starting from the same initial state s_0. This conversely prohibits learning in a productive environment due to the fact that deleting a resource for instance might be nonreversible. Therefore, there is the necessity of a clone with the same semantics and behavior like the productive environment, which can be reset to the initial state after each episode of learning.

Apart from this, the research focus has been on single-agent environments so far. There has been no investigation on how to adapt the approach to multi-agent scenarios.

Finally, we have the reward function whose definition depends on the applications goal (i.e. the desired target state which should be reached by the agent). In section III, we gave a generic example how a reward function can be defined if the agent should learn an optimal policy in terms of *as few service calls as necessary* to achieve a certain goal. In section IV, we have shown how such a reward function can be implemented in the form of an observer. It remains open whether such a generic approach is applicable in real-word scenarios (especially in complex service environments) and whether there are further generic approaches which can be derived from different use cases and interoperability scenarios. As a long-term goal, a developer should be able to choose from several generic best practice approaches and derive an individual reward function without much effort. As part of our further research, we want to investigate how reward functions in the context of interoperability can be defined generically.

VI. Conclusion

In this paper, we introduced an approach how a REST service can be modelled as a Markov Decision Model (MDP). Moreover, a Q-Learning algorithm has been adapted to the properties of REST such that an agent is able to learn the semantics of a given REST service as well as an optimal sequence of service calls for achieving an application specific goal. It turned out, that the REST constraint HATEOAS eases the adaptation as it urges the service to be implemented hyper-driven and as a state machine. However, during modelling and adaptation we have identified some issues in our approach which might complicate a practical use in a real-world scenario. Nevertheless, we were able to show that it is possible, to address semantic interoperability issues with techniques from the field of AI. In our further research, we plan to extend this approach on other protocols as well as on other levels of interoperability.

References

[1] P. Fältström, "Market-driven Challenges to Open Internet Standards," *Global Commission on Internet Governance Paper Series, Centre for International Governance Innovation (CIGI)*, no. 33, May 2016.

[2] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, April 2012.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, June 2015.

[4] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in Internet of Things: Taxonomies and Open Challenges," *Mobile Networks and Applications*, July 2018.

[5] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A Survey on Facilities for Experimental Internet of Things Research," *IEEE Communications Magazine*, vol. 49, pp. 58–67, November 2011.

[6] B. Ahlgren, M. Hidell, and E. C.-H. Ngai, "Internet of Things for Smart Cities: Interoperability and Open Data," *IEEE Internet Computing*, vol. 20, pp. 52–56, December 2016.

[7] N. L. Tsilias, "Open Innovation and Interoperability," in *Opening standards: The global politics of interoperability*, L. DeNardis, Ed. Cambridge, MA, USA: MIT Press, 2011, pp. 97–117.

[8] G. Hatzivasilis, I. G. Askoxylakis, G. Alexandris, D. Anicic, A. Bröring, V. Kulkarni, K. Fysarakis, and G. Spanoudakis, "The Interoperability of Things: Interoperable solutions as an enabler for IoT and Web 3.0," in *23rd IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD 2018, Barcelona, Spain*, September 2018, pp. 1–7.

[9] E. Wilde, "Putting Things to REST," School of Information UC Berkley, California, USA, Tech. Rep. UCB iSchool Report 2007-015, November 2007.

[10] A. L. Lemos, F. Daniel, and B. Benatallah, "Web Service Composition: A Survey of Techniques and Tools," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 33:1–33:41, December 2015.

[11] S. Nagowah, H. Ben Sta, and B. Gobin, "An Overview of Semantic Interoperability Ontologies and Frameworks for IoT," in *Sixth International Conference on Enterprise Systems (ES), Limasol, Cyprus*, October 2018, pp. 82–89.

[12] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[13] K. Sangsanit, W. Kurutach, and S. Phoomvuthisarn, "REST Web Service Composition: A Survey of Automation and Techniques," in *International Conference on Information Networking (ICOIN), Chiang Mai, Thailand*, April 2018, pp. 116–121.

[14] M. Tang and L. Ai, "A Hybrid Genetic Algorithm for the Optimal Constrained Web Service Selection Problem in Web Service Composition," in *IEEE Congress on Evolutionary Computation, Barcelona, Spain*, July 2010, pp. 1–8.

[15] S.-C. Oh, D. Lee, and S. Kumara, "Effective Web Service Composition in Diverse and Large-Scale Service Networks," *IEEE Transactions on Services Computing*, vol. 1, pp. 15–32, January-March 2008.

[16] M. Bennara, M. Mrissa, and Y. Amghar, "An Approach for Composing RESTful Linked Services on the Web," in *Proceedings of the 23rd International Conference on World Wide Web, Seoul, Korea*, April 2014, pp. 977–982.

[17] R. Verborgh, T. Steiner, D. Van Deursen, J. De Roo, R. Van de Walle, and J. Gabarró Vallés, "Description and Interaction of RESTful Services for Automatic Discovery and Execution," in *Proceedings of the FTRA 2011 International Workshop on Advanced Future Multimedia Services (AFMS 2011), Jeju, Korea*, December 2011, pp. 12–15.

[18] H. Wang, X. Zhou, X. Zhou, W. Liu, and W. Li, "Adaptive and Dynamic Service Composition Using Q-Learning," in *22th International Conference on Tools with Artificial Intelligence, IEEE, Arras, France*, October 2010, pp. 145–152.

[19] H. Wang, Q. Wu, X. Chen, Q. Yu, Z. Zheng, and A. Bouguettaya, "Adaptive and Dynamic Service Composition via Multi-agent Reinforcement Learning," *2014 IEEE International Conference on Web Services, Anchorage, AK, USA*, pp. 447–454, June 2014.

[20] H. Wang, X. Chen, Q. Wu, Q. Yu, X. Hu, Z. Zheng, and A. Bouguettaya, "Integrating Reinforcement Learning with Multi-Agent Techniques for Adaptive Service Composition," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 12, pp. 1–42, May 2017.

[21] L.-B. Feng, M. Obayashi, T. Kuremoto, K. Kobayashi, and S. Watanabe, "QoS optimization for web services composition based on reinforcement learning," *International Journal of Innovative Computing, Information and Control*, vol. 9, pp. 2361–2376, June 2013.

[22] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, USA, 2000.

[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[24] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989.

[25] M. Nottingham, "Web Linking," Internet Requests for Comments, RFC Editor, RFC 8288, October 2017.

[26] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," Internet Requests for Comments, RFC Editor, RFC 7231, June 2014.