Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 137 (2018) 78–89

Procedia Computer Science

www.elsevier.com/locate/procedia

SEMANTiCS 2018 – 14th International Conference on Semantic Systems

Using a Semantic Simulation Framework for Teaching Machine Learning Agents

Nicole Merkle[a], Stefan Zander[b]

[a] *FZI Forschungszentrum Informatik am KIT, Haid-und-Neu-Str. 10-14, Karlsruhe 76131, Germany*
[b] *University of Applied Sciences Darmstadt, Schöfferstrasse 8B, Darmstadt 64295, Germany*

Abstract

Autonomous virtual agents that operate in complex IoT environments and apply machine learning algorithms face two fundamental challenges: (i) they usually lack sufficient start-up knowledge and (ii) hence are incapable to adequately adjust their internal knowledge base and decision-making policies during runtime to meet specific user requirements and preferences. This is problematic in Ambient Assisted Living (AAL) and Health-Care (HC) scenarios, since an agent has to expediently operate from the beginning of its lifecycle and adequately address the target users' needs; without prior user and environmental knowledge, this is not possible. The presented approach addresses these problems by providing a semantic use-case simulation framework that can be tailored to specific AAL and HC use cases. It builds upon a semantic knowledge representation framework to simulate device events and user activities based on semantic task and ambient descriptions. Through such a simulated environment, agents are provided with the ability to learn the best matching actions and to adjust their policies based on generated datasets. We demonstrate the practical applicability of the simulation framework through the evaluation of the chronic kidney disease pathway from the vCare EC project. Thereby, we proof that an agent that uses reinforcement learning (RL) is able to improve its performance during and after the training and thus makes optimal (activity) recommendations to a prospective patient.

Keywords: Agent-Based Systems; Semantic Web; Healthcare; Context-Awareness; Reinforcement Learning;

1. Introduction

At the beginning of an agent's lifecycle, i.e., when it is first deployed in a real user environment, an agent's internal knowledge base contains only default knowledge without specific, individualized, and adapted knowledge. It needs

* N. Merkle. Tel.: +49-721-9654-896 ; fax: +49-721-9654-897.
E-mail address: merkle@fzi.de

1877-0509
10.1016/j.procs.2018.09.008

to be programmed and taught to perform its assistive tasks in a sufficient manner. This problem is called *cold-start problem*. From a developer perspective, it is cumbersome to re-program or re-align an agent's operational logic and internal decision-making processes for every specific task and context. Moreover, it is impossible to consider all situations and exceptions, which might occur during an agent's lifecycle at its design time.

Especially in a complex environment with obstacles and diverse IoT[1] devices (sensors and actors), an agent has to handle high-dimensional, multi-modal observations and requires validated knowledge about how to react to these observations adequately. This high-dimensional observation space makes it necessary to apply machine learning (ML) approaches (e.g. neural networks), because in ML there are a bunch of algorithms, which can handle a high-dimensional feature space and approximate non-linear distributed data (e.g. images with a high resolution). Moreover, knowledge about relevant things in the environment is required in order to enable the agent to interpret the observations and to determine, which actions it is able to perform. This ability depends on the availability of IoT devices during its runtime as well as to other requirements such as the appropriate internal logic, compatible APIs, the handling of device and data heterogeneities. The agent needs to know, which functionality a device provides and how to evoke this functionality. In our previous work [8, 9], we addressed these issues by providing a WoT server implementation that allows the communication between IoT devices and agents. In addition, the agent requires to know, which effect (positive or negative) a performed action has for itself, the user and its surrounding environment. By these effects, the agent shall be able to learn the user's requested actions. Therefore, we provide a simulation environment whose meta-model is being described by a semantic knowledge representation framework. We provide the meta-model based on health-care guidelines[2] and clinical pathway[3] recommendations to represent patient profiles, vital parameter observations, activities and their effects. The advantage of this meta-model is that it has to be created once and provides the basis for the simulator to generate realistic datasets, which are reproducing these guidelines. Thereby, the research challenge is to provide a simulator, which is able to reproduce different patient profiles and to combine these patient profiles and their disease patterns. Moreover the simulator framework requires to consider the guideline activities and to generate them for heterogeneous patient instances. This is a difficult task, since the simulator framework requires to consider the meta-model in its data generation process and intelligently transform and relate this meta-knowledge with the generated instances. The meta-model provides all necessary criteria, which the simulator needs to heed in its data generation process.

If a strategy (policy) is learned by the agent, it can share its learned policy with other agents in a multi-agent system by a central database or semantic collaboration framework e.g. Semantic MediaWiki (SMW). In this paper, we lay our focus on the simulation framework and the representation of the meta-model. Moreover, we apply a ML approach (Deep Q-Learning (DQN) in order to train the agents for learning their policies. In this way, we overcome the problem of missing training data in the healthcare domain and coherently the cold-start problem. Moreover, the agents benefit of each others experience and provide their services adequately to the user. The remainder of this work is structured as follows. Section 2 discusses related work, considering agent-based systems in different domains. Section 3 presents an example use case (Chronical Kidney disease pathway (CKDPathway)) from the vCare[4] project that has been addressed by our approach in order to evaluate the feasibility and performance of our approach. Section 4 and 5 outline the approach in detail and demonstrate the technical feasibility of the approach through a proof-of-concept scenario. Section 6 summarizes results and benefits of this work.

2. Related Work

The cold-start problem is a well-known problem in many domains (e.g. agent-based and recommender systems). (e.g. [17, 5, 19, 7]). Numerous approaches have been developed for handling *item* and *user cold start* problems: Active Learning, a special field of semi-supervised machine learning, has been elaborated by several authors (cf. [18, 2, 16]). Those approaches aim at evaluating the usefulness of data points in order to improve data quantity and recommender system performance. Multi-agent systems often embody strategies for data exchange and collaboration (e.g. [14, 4]) to

[1] Internet of Things
[2] https://www.acponline.org/clinical-information/guidelines
[3] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4561460/
[4] http://vcare-project.eu

allow agents to learn from other agents in order to provide appropriate assistance in new and unknown environments and contexts. A third type are hybrid approaches, e.g. combining content-based matching approaches and collaborative filtering [17] or association rules and clustering techniques[1]. Those approaches revealed to be helpful in handling the cold-start problem by deducing similarity indicators from content-based characteristics. In order to address the lack of user profiles, Yahoo! research developed a collective learning representation framework to tackle both the item cold start and the user cold start by using matrix factorization techniques (cf. [7]) such as alternating least squares and multiplicative updates [6].

As the previous paragraph indicates, many approaches have been proposed to overcome cold-start problems, but the deployment of semantic technologies and description frameworks together with semantic simulation frameworks have received only sparse attention. Foundational to those approaches is the work of Middleton et al. [10], which first studied the synergies between recommender systems and semantic knowledge structures such as ontologies. Other works such as Nouali et al. [13] demonstrated the increase of precision, coverage and quality of recommender systems by semantically enhanced descriptions of user and resource information. A study about general semantic recommender systems has been published by [15]. Although the positive effects of semantic technologies on recommender systems are broadly known (cf. [22]), many recent approaches (e.g. [3, 12, 21]) started to analyse the extent to which recommender systems can be enhanced using Linked Open Data (cf. [12]). However, a semantic simulation framework comparable to the one in this work was not specifically presented. A very similar approach that addresses the sparsity and scarcity problem was developed by Thanh-Tai et al. [20]. In contrast to our approach that, based on guidelines, generates new data, their approach uses a semantic model to generate similarity data for a given original dataset.

All considered related work show the relevance of the cold-start problem within agent-based systems for solving real-world problems in different domains. In particular, the referenced approaches provide modelling and design techniques as well as interaction languages and protocols. However, regarding the universality of the proposed approaches, mostly, the presented agent-based systems are restricted by their model representation and their design for specific domains. Moreover, the on-the-fly integration of new tasks as well as the self-programming of agents by ML is not sufficiently discussed in the considered related work. We try to close these gaps by providing an agent-based framework that combines semantic technologies and DQN in order to enable agents to learn by themselves how to solve various given tasks and to adapt to context-specific data. Moreover, the framework presented in this work allows agents to cooperate through sharing and reusing their strategies (policies) via a semantic knowledge exchange platform. In this way, the agents are enabled to re-align their behaviour without the intervention of a developer.

3. Example Use Case – Chronic Kidney Disease Pathway

This section describes a clinical pathway (CP) use case from the health-care domain that embodies a certain degree of complexity and adequacy for evaluating our approach. In general, a CP specifies the treatment process of a patient subsequent to a certain disease diagnosis; its objective is, in general, the patient's full recovery. In the ideal case, this objective is achieved by conducting certain recommended activities (e.g. exercises, dietary instructions, cessations, etc). A CP can be rather complex and usually involves an experienced physician in order to make appropriate decisions and examination recommendations. In some cases, several activity recommendations are appropriate for an observed state, so that the decision making might be even more difficult. If the physician's decision making tasks are transmitted to a ML agent, then we have to consider, which kind of observations (e.g. vital sign parameters) are relevant in order to make adequate decisions. Moreover, the agent requires to follow certain guideline rules that describe relevant observations related to states and possible activities that can be performed in order to support the patient's recovery.

To illustrate the process of learning of appropriate activity recommendations for a CP, we discuss the *chronic kidney disease pathway (CKDPathway)*[5] along which we demonstrate the benefits of our approach. It consists of different states and activity recommendations that are based on clinical guidelines. The CKDPathway starts with the diagnosis of chronic kidney disease (CKD) and ends with the achievement of appropriate target states (e.g. A1C

[5] http://ckdpathway.ca

value < 7%, BMI[6]: 18.5 - 25, blood pressure: 130/80mmgH, etc). These target states prescribe the objectives of the CP and implicitly provide a measure for the patient recovery.

For the diagnosis of CKD, two different vital sign parameters (*ACR* and *eGFR* value) are tested. If CKD is diagnosed, a second examination of the given vital signs is indicated together with an urine analysis. Considering the CKDPathway, we determined nineteen different states (i.e. *UserHasCKDRisk, UserHasNoCKDRisk, UserHasCKDWithDiabetes, UserHasNoCKD, TargetBloodpressure, TargetA1C, UserHasCKDWithoutDiabetes*, etc.) and thirty-seven activity recommendations (i.e. *FluidIntakeRegulation, TestACR, TestEGFR*, etc.). All of these states are determined by given guideline rules. If the rule conditions (e.g. ACR >= 3mg/mmol) are met, then an agent can recognize the appropriate state and decide the appropriate recommendations based on related activities. However, if there are more than one activities related to a state, the challenge for the agent is to decide which activity is the best one for the patient. CPs do not provide this information and that is the reason why our approach introduces the concepts of *rewards* and *punishments*. Based on the *effects* of an activity, the domain expert decides the rewards or punishments that are assigned to a state-action pair. Therefore, if an activity effect is desired, the agent will receive a reward for recommending this activity, otherwise it will receive a punishment. How this is realized by the proposed approach is discussed in Section 4. For more in-depth details about this example CP, we refer to the appropriate website[7].

4. Approach

The proposed approach utilizes different methodologies of ML and semantic web technologies and is discussed in the next sections. First, we give an overview regarding the architecture and functional sequence. Then in the subsections, we go into detail and show how the semantic representation of the agent's environment looks like. Furthermore, we describe in detail the simulator, which utilizes the semantic representation in order to simulate events and state changes in the environment. The agent learns by means of this simulation appropriate policies for requested tasks (e.g. activity recommendations). A policy is trained by a deep neural network (DNN), recommending an action according to a certain state. A state thereby is described by sensed observations because every performed action of an agent has an effect to the patient and the states of the environment. It is important to take into consideration that every state is more or less preferred. In order to formalize this, we use—according to the RL approach—reward or penalty values for every performed action. States that are assessed by the domain expert as good, get assigned a reward value of one. Neutral assessed states get a value of zero and negative assessed states a value of minus one. We restrict the range of reward values because we want to avoid scaling issues.

Figure 1 depicts—ordered by numbers—the entire architecture of our approach. 1) In the first step, a domain expert creates via SMW[8] a semantic representation of the environment. This representation comprises the agent profile, which defines the actions that an agent can perform. Moreover, it sets up, for which IoT devices an agent requires to subscribe. For this reason, the domain expert defines IoT device-specifications of devices, which are integrated in the environment and are relevant for the agent's observations and actions. After the ambient specification is set up, the user profile of the target user has to be specified. This profile comprises of demographic data (e.g. gender, age, preferences, etc.) and a specification of user capabilities (motoric or mental) as well as impairments. Once the semantic ambient-, agent- and user profiles are created in the SMW, the domain expert creates task specifications. The task specification consists of linked states, actions and parameters (e.g. discount value, learning rate) for the RL approach. SMW generates out of this specifications light-weight RDF(S) instances, which can be queried by running agents via the related SPARQL endpoint. 2) An agent performs at the beginning randomly an action, which is predefined in its profile. 3) This action is sent to the simulator. 4) The simulator requests subsequently from the knowledge representation framework the effects of the performed action. These effects represent new assumed observations (state changes) in the environment. Every state change is triggered by certain effects. 5) If the current state is determined by the simulator, it reasons by the effect's rules, the state's reward value in order to send it to the agent. 6) The agent stores this reward value and additional information in a replay memory and takes a batch of the data out of this replay memory in order to compute by a DQN the future expected reward for an action, performed in a certain state. Every

[6] Body Mass Index
[7] http://ckdpathway.ca
[8] see https://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki

performed action leads to a new state from where the agent performs a new action until the agent achieves a goal state. The goal state is predefined by the domain expert via the task description. The agent's learning iteration from a starting state to a goal state is called an episode. Usually, an agent performs several thousand episodes starting from every state until the output values of the DQN converge. If this is the case, the DQN is trained for a certain task and can be used by the agents in order to solve this task and provide adequate services.

7) In the next step, the agent stores the trained DQN model with its weights and parameters. 8) The location (storage path) of the trained DQN model is published by the agent via the task representation in SMW. However, it is also possible to store/publish the data in other data formats, depending on the requirements of the agent. We decided to use SMW because it provides an API, which can be utilized by agents in order to create/update/delete new semantically annotated wiki pages. The RDF(S) representation of these wiki pages is then accessible via a SPARQL endpoint. Moreover, SMW provides templates, which the agent can fill-out with the appropriate entry values, while SMW transforms these template values automatically via annotations into RDF(S) graphs.

In our approach, SMW synchronizes its entries with the RDF4J[9] triple store and provides access via an integrated SPARQL endpoint. 9) Via this endpoint, local network agents are able to retrieve the published policies for the given tasks. As soon as the agent starts to make real (no simulated) observations in the environment and determines its or the user's state, it creates SPARQL queries to search for a matching policy, referenced in the SMW. After the policy retrieval and utilization of the DNN, it performs the recommended action to maximize its accumulated rewards.

The idea is that agents observe their environment by subscribing for device events to a *Web of Things (WoT)* server. The WoT forwards occurring device events to the subscribers of those events and sends actions from the subscribers to the devices in order to control them. 10) Moreover, the WoT server stores every occurring event and action as datasets in a datastore. A dataset consists of a timestamp, occurred device events, performed actions and the sender (e.g. IoT sensor) of an event and action. 11) Since we have here different features and data representations, a preprocessor component queries frequently from the datastore a new batch of datasets and transforms their representation by means of the semantic feature representation into a numerical one. In addition, it takes user-specific data (e.g. health-status, preferences, demographic data) and transforms these as well into a numerical representation, 12) so that all together can be processed in a vectorized representation by a deep neural network (DNN). This transformation and adjustment is necessary in order to personalize the agent's provided services, since it allows to consider additional context history data (e.g. user profile, conducted actions, sensor events, etc). The outcome of the DNN is the most likely action to perform under the given circumstances. 13) According to this outcome, the agent updates the related published policy (DNN), so that the policy represents the best matching solution for a task (problem). However, we do not discuss how this is implemented because this goes beyond the scope of this paper.

4.1. The Semantic Knowledge Representation of the Agent's World

In order to allow the agent to process the given information, our approach provides a semantic model that represents the agent and the *world* in which it has to act. The agent's world consists of the agent description and of tasks that it has to conduct in order to solve a problem.

In the following paragraphs, we present the different semantic elements of an agent's world.

The Task Description provides for the agent the problem, which it has to solve. A task can be considered as a use case. Every task description consists of different states and actions. In addition, every task has a goal state, which is a special kind of a state and terminates the task. Furthermore, every task has a processing status that shows the agent, if a task is *OPEN*, *IN PROGRESS* or *DONE*. An agent requests just open tasks in order to train policies (strategies) for that task. During the computation of the policies, the agent sets the process status of the task to *IN PROGRESS*. This is necessary in order to avoid in a multi-agent system, that other agents compute in parallel policies for that task. After the policies are trained, the agent marks the task as *DONE* in order to show that it is no longer required to train a strategy for that task. Tasks can be sequential, so they can be linked to previous and next tasks in order to structure several tasks. Axiom 1 shows the class definition of a task.

The State consists of guideline rules, which define the conditions for a state existence. These conditions are related to observations, sensed by IoT devices (wearable and stationary) in the environment and on the user. The guideline

[9] see: http://rdf4j.org/

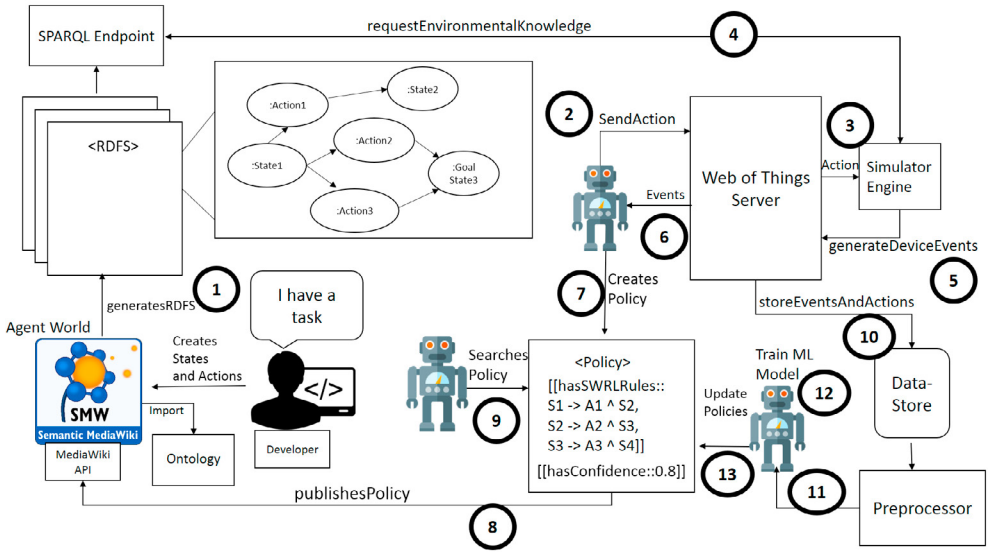


Fig. 1: The system architecture.

rules are depending on the domain. For instance for the health-care domain, we took medical guidelines that specify ideal values for certain vital sign parameters. Axiom 2 shows an example of such a rule taken from the CKDPathway, which determines a state called *UserHasCKD*.

$$\begin{aligned} \text{Task} \sqsubseteq \exists\, \text{hasState.State} \\ \sqcap\ \text{hasGoal.State} \ \sqcap\ \text{hasProcessingStatus.}\{\text{OPEN}\} \\ \sqcup\ \text{hasProcessingStatus.}\{\text{INPROGRESS}\} \\ \sqcup\ \text{hasProcessingStatus.}\{\text{DONE}\} \\ \sqcap\ \text{hasPreviousTask.Task} \ \sqcup\ \text{hasNextTask.Task} \end{aligned} \tag{1}$$

$$\begin{aligned} \text{ObservationFeature(eGFR)} \ \sqcap\ \text{Agent(?a)} \\ \sqcap\ \text{ObservationFeature(ACR)} \\ \sqcap\ \text{hasValue(eGFR, ?egfr)} \ \sqcap\ \text{hasValue(ACR, ?acr)} \\ \sqcap\ \text{lessThan(?egfr, 60)} \ \sqcap\ \text{greaterThanOrEqual(?acr, 3)} \\ \Rightarrow\ \text{isInState(?a, UserHasCKD)} \end{aligned} \tag{2}$$

The rule expresses by the given conditions if a person has CKD. The antecedent of the rule defines that there are observation feature instances of *eGFR* and *ACR*. The mentioned observation features have certain values that can be located in a certain numerical range. If the antecedent holds then the agent can reason that a person has CKD or not. A *State* class definition is expressed by Axiom 7. A state allows to perform certain *Actions* and can be either a goal state or not. Every state is linked to actions that are possible in the given state. As already discussed, every state is described by rule conditions that are related to observation features. Since a state is observed by (wearable and stationary) devices, it is linked to these devices. An agent is then enabled to check for all devices, which are relevant for observing a state.

The Agent Profile is requested by an agent at the beginning of its lifecycle. The agent requires this profile information in order to identify relevant devices for its subscription at the WoT server. Furthermore, the agent retrieves indirectly by the profile, which actions it is able to perform. The actions are also implicitly given by the device functionality, which is explicitly defined by the IoT device description. Since every ML algorithm needs training parameters (e.g. discount factor, learning rate, momentum, etc.), the agent profile also provides these kind of parameters. Axiom 3 illustrates the class definition of an agent profile. In order to get the related device actions, the agent requires to query by SPARQL for the devices' functionality. Therefore, in the agent profile, a SPARQL query is defined. As soon as the possible actions are retrieved, the agent performs randomly one of the given actions and sends it to the simulator that looks-up its effects, assigns based on the effect a reward value and sends it as answer to the agent.

The semantic policy description is autonomously generated by the agent after the computation of the task related policies. A policy class is defined in Axiom 4.

$$\text{Agent} \sqsubseteq \exists\ \text{subscribeForDevice.Device} \sqcap \text{hasSparqlQuery.Query} \sqcap \text{hasLearningParameter.Parameter} \quad (3)$$

$$\text{Policy} \sqsubseteq \exists\ \text{hasState.State} \sqcap \text{hasRule.Rule} \sqcup \text{hasDNN.DNN} \sqcap \text{hasGoal.State} \quad (4)$$

A policy instance is linked to task specific states. Every policy instance requires a goal state in order to terminate a task. Either the rules or the DNN representation purport the actions, which the agent has to perform depending on the agent's observed state. Listing 6 shows a policy rule that dictates the agent's next action depending on the given observed state.

The IoT Device Description provides information about the device's functionality (actions), its evoked events and its properties. An IoT device can have different state ranges, while these state ranges belong to a datatype property (e.g. string, boolean, number, etc). Moreover, every device is located somewhere in the environment, depending on if it is a wearable or stationary device. Locations are room instances or persons who are wearing the device on their body. Axiom 5 depicts the class definition of an IoT device description. The detailed description of IoT device representations and the WoT approach are discussed in our previous works [8, 9].

$$\text{Device} \sqsubseteq \exists\ \text{hasFunctionality.Action} \sqcap \text{hasDataType.owl : DatatypeProperty} \sqcap \text{hasStateRange.Range} \sqcap \text{isInRoom.Room} \sqcap \text{isOfType.Wearable} \sqcup \text{isOfType.Stationary} \quad (5)$$

$$\begin{aligned} &\text{State(UserHasCKDRisk)} \sqcap \text{Agent(?a)} \\ &\sqcap \text{IsInState(?a, UserHasCKDRisk)} \\ &\sqcap \text{Action(TesteGFR)} \sqcap \text{Action(TestACR)} \\ &\Rightarrow \text{HasOptimalAction(UserHasCKDRisk, TesteGFR)} \\ &\sqcap \text{HasOptimalAction(UserHasCKDRisk, TestACR)} \end{aligned} \quad (6)$$

The example of rule 6 illustrates that a state *UserHasCKDRisk*, two actions (*TesteGFR, TestACR*) and an agent instance are given. The agent observes by state rules that it is in a certain state (*UserHasCKDRisk)*. If the antecedent holds, then the agent can imply the optimal action, which is in this case, *TesteGFR and TestACR*. Every policy rule is built by the agent in the same structure and can be requested via the SPARQL endpoint depending on the observed states. However, the agent has also the opportunity to store instead of rules, the representation (e.g. weights and parameters i.e. learning rate, discount factor, etc.) of the trained DNN. The reason for using DNNs models instead of rules is that there are often complex tasks with high-dimensional sensor state spaces, which cannot be transformed easily into rule representations. The DNN representation defines the characteristics of the DNN (e.g. weights of every layer, layer size, activation functions, etc.) and is utilized by the agents as activity predictor in order to decide for the next action. Axiom 10 defines the DNN class representation.

The User Profile provides personal data about the target user. It consists of demographic data (e.g. gender, age, origin, language, family status, etc.), preferences, social activities and medical records of the patient. The properties of the user profile data are later used for the DNN as input. Therefore, the patient profile data has to be transferred by one-hot-encoding into a numerical vector representation. Together with the patient data also state events serve as input data for the DNN. In this way, we achieve a personalized adjustment of policies during the runtime of the agent system.

The Virtual Influencer represents a virtual sensor within the simulation environment that changes dynamically varying observation values. The virtual influencer instance is especially for cases required in which an agent does not directly influence observation features of the given states. In most of the cases, sensors of the environment provide observations that are influenced by outside conditions. Considering this, the virtual influencer produces these outside influences and introduces thereby some uncertainty into the simulation environment. This is required in order to make the simulation realistic. For instance, the position of the patient or the vital parameters might change during runtime, independent of the direct agent activities. Axiom 9 specifies the virtual influencer class. Virtual influencers are related to effects of the environment, which implies that certain observation features can be changed by the virtual influcencers. Therefore, every virtual influencer is linked to an effect.

An Action can be performed by an agent via an IoT device. A set of actions are allowed to the agent, depending on the task and the installed IoT devices. Every action can have multiple effects to the current state of the environment. Axiom 8 defines the action class:

$$\text{State} \sqsubseteq \exists \text{hasAction.Action} \sqcap \text{isGoal.Bool} \sqcap \text{hasObservationFeature.ObservationFeature} \sqcap \text{hasCondition.Condition} \sqcap \text{hasRelatedDevice.Device} \quad (7)$$

$$\text{Action} \sqsubseteq \exists \text{hasEffect.Effect} \quad (8)$$

$$\text{VirtualInfluencer} \sqsubseteq \exists \text{hasEffect.Effect} \quad (9)$$

An Effect describes the impact of an (agent) activity or virtual influencer to the observation features of the environment. Every effect causes a change of observations. Moreover, an effect allows the simulator to assign rewards or punishments to state-action combinations, because every effect has to be considered together with a given state. For instance, if a patient has the state *Overweight* then the effect *IncreaseBMI* might be unwanted. In turn, if the patient has *Underweight*, then the effect of an increased BMI might lead to a positive reward because the objective is to regulate the patients weight. Therefore, it is not simply possible to asses effects on their own. They always have to be assessed in relation with the current state. Axiom 11 illustrates an effect representation.

$$\text{DNN} \sqsubseteq \exists \text{hasActivationFunction.Function} \sqcap \text{hasHiddenLayerSize.}(= 3) \sqcap \text{hasLayerType.LayerType} \sqcap \text{hasOutputSize.}(>= 1) \sqcap \text{hasInputSize.}(> 1) \quad (10)$$

$$\text{Effect} \sqsubseteq \exists \text{hasImpact.Impact} \sqcap \text{hasObservationFeature.ObservationFeature} \sqcap \text{hasRangeImpact.Decimal} \sqcap \text{hasRule.String} \quad (11)$$

Every effect has an impact (e.g increase, decrease, convert) to exactly one observation feature value. The impact range is a decimal number that specifies in which range the observation feature value can be changed by the appropriate effect. The simulator adds or subtracts this value range from the appropriate observation feature if the effect occurs due to a performed activity. A *convert* effect allows to switch values from zero to one and vice versa.

4.2. The Simulator

The simulator component provides for an agent a virtual simulation environment based on the presented model representation. As soon as the agent starts to send randomly actions to the simulator, it queries via SPARQL from the local SPARQL endpoint the linked effect(s) of a performed action. Hence, the action is utilized for retrieving the related effect. The effect indicates then how the current state that is represented by a numerical feature vector, has to be changed. As already discussed, every vector element represents a feature and the appropriate effect either increases, decreases or converts the feature value. The effect(s) are related to an impact (i.e. *Increase, Decrease, Convert*) and two rules (SPARQL construct queries) that indicate when an action gets a reward or punishment depending on the given state. Therefore, the simulator adds the current state representation as RDF triples to the current A-Box representation of the simulation. Subsequently, it performs the given construct query in order to retrieve the matching rewards. The following listing depicts an example construct query of an effect:

```
CONSTRUCT {:IncreaseBMI :hasReward "-1.0"^^xsd:double}
WHERE {
  ?agent :performsActivity ?activity.
  ?activity :hasEffect :IncreaseBMI.
  ?agent :isInState :PatientHasOverweight.}
```

The SPARQL construct query prescribes that if the patient has overweight and performs an activity that increases the BMI value, then the agent gets for recommending this activity a punishment of -1.0. On the one hand, an action influences directly by its related effects the simulation states and on the other hand, the effect helps to asses the appropriate action, considering the current context (healthstate) of the patient. As soon as the reward is determined, the simulator sends the new state representation together with the appropriate reward to the agent. The agent decides, based on the collected rewards and the new state, the next activity recommendation. The reward value of a state is important, since the agent utilizes these reward values for computing the quality value (q-value) for a state and its related action. A q-value represents the computed highest expected future reward of a state and its related action. If the next state is also retrieved, the agent can continue its computation from there until a goal state is achieved. The

interaction between the agent and the simulator continues in several thousand episodes until the agent detects that the q-values of the state-action pairs are converging. If this happens, the agent knows that the learning phase can be stopped and sends a message that indicates to the simulator the end of the training.

4.3. Deep Q-Learning for Action Policies

The agents in our approach use DQN for learning the policies of task descriptions as proposed by DeepMind[10] in [11]. We decided to apply DQNs because they allow to process high-dimensional input data and provide good results in training agents in other domains. Since the environment in that the agents act, can have a high-dimensional sensor state space (e.g. pixels in image processing), it makes sense to use DQNs. In particular, DQNs are applied for learning to play games (e.g. ATARI, Go, etc). However, a DQN can also be used for other use cases and settings as in our case for IoT environments. Usually, the environment of an agent is stochastic, which means that the next occurring state is random. For this reason, we provide by the semantic task description (see Section 4.1) a Markov-Decision Process (MDP)[11], which defines the possible states, their rewards and actions of a task. In this way, we get a finite sequence of states, rewards and actions. Every task (episode) terminates at the goal state. The agent learns based on this task description the appropriate policies for a task.

In our approach, the agents implement a DQN that takes as input the observed sensor states, which are transformed beforehand into a scaled numerical representation. The output of the DQN consists of q-values for every possible action. A q-value indicates the action with the highest expected reward value. After the network is trained, the agent decides in every single state for the action with the maximum q-value (output). The DQN consists of three hidden layers. All hidden layers in the DQN are fully connected. As activation function for every layer, we apply—according to DeepMind's proposed DQN—rectified linear units (ReLU). However, we do not use convolutional layers, because in the current state, we have no image processing in our use cases. However, if image processing should be required, the DQN can be adjusted depending on its inputs. The squared error loss function, which is used in Deep Mind's DQN is depicted in Equation 12. It is used to adjust the q-value outputs iteratively by backpropagation until they converge.

$$L = \frac{1}{2}[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]^2 \tag{12}$$

The first part of the loss function (until to the minus sign), is the target function. The prediction value *Q(s, a)* is given, after the minus operator. The target function computes the q-value for *State s* and *Action a*, considering the maximum future reward of the next state and the next state's action. The prediction value is the computed q-value of the current state and action. By computing the difference between this both parts of the loss function, the algorithm determines the deviation of the prediction value from the target value. The DQN is trained in the following way: 1) The new state is `feedforwarded` through the DQN in order to get q-values for all actions. 2) The next occurring state is as well `feedforwarded` in order to calculate the target value, which is the maximum q-value of the next state and its action. 3) The maximum q-value of all outputs of step 2 is selected as target value. All other actions are set to the target value as computed in step 1. 4) In the last training step, the weights of the DQN are updated by backpropagation. In order to accelerate the training process, we use—as proposed by DeepMind—experience replay. The state, action, reward and nextstate are stored in a data structure called *replay memory*. Randomly stored samples of the experience replay memory are taken in order to train the DQN. The advantage of this approach is that the samples are not similar since the sample batches are randomly taken from the replay memory. In order to avoid the *Exploration-Exploitation Dilemma*, the agent decides by a greedy strategy, if it performs randomly an action or if it selects the action with the maximum q-value. Therefore, it chooses random actions by a probability given by an epsilon value. Otherwise, it selects the action with the maximum q-value. The epsilon value is decreased over time from 1.0 to 0.1 since the q-values converge and provide the best actions to perform in a certain state. In this case, the exploitation of the q-values makes more sense. After the policy page—with the link to the trained model—is created and published by the agent, its RDF(S) representation is accessible by other agents via the local SPARQL endpoint. If an agent wants to retrieve a matching policy for a certain observed state, the agent requests it by SPARQL queries. In this way, SMW is used as collaboration platform for the proposed multi-agent system. Agents are enabled to share and retrieve policies for every task that is provided in a semantic representation by the domain expert.

[10] https://deepmind.com/

[11] A MDP means that the probability of a state just depends on the previous state and the performed action, but not on preceding states or actions.

5. Proof of Concept

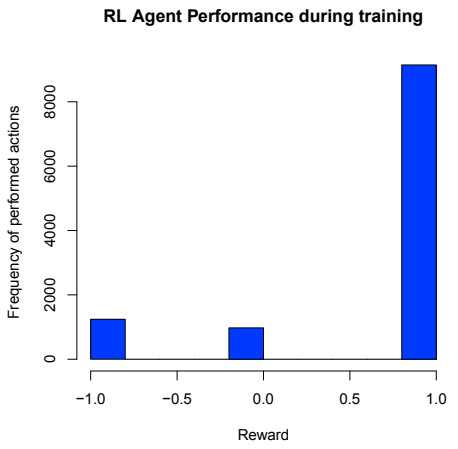
The PoC of the presented approach evaluates the introduced CKDPathway use case. Our objective is to demonstrate that the RL agent is able to learn by the simulator the best matching activities. An indicator that the agent has learned the best activities for the given states, is that it increases it reward contingent. The more rewards and less punishments it gets by every activity recommendation, the better is its performance. The setting of the PoC is as follows: First, we create a use case instance that represents the CKDPathway. The agent has the task to observe certain vital parameters (e.g. ACR, eGFR, Hematuria, Potassium) of a virtual patient. Based on these observations, the agent has to recommend activities that the patient shall conduct. Every activity has a certain effect to the patient's vital parameters. These effects can increase, decrease or convert the observation values. Therefore, every state can just be evaluated together with the effects of an activity. The result of an evaluation is either a reward (1.0) or a punishment (-1.0). If the effect is assessed as neutral, the reward value is zero. The actions, which the agent can recommend are: *SportExercise, LowPotassiumSodiumDiet, SmokingCessation, RegulateFluidIntake, ARBIntake, StatinIntake, AntiplateletIntake*. These actions are defined in the CKDPathway instance.

For the evaluation of the given task, we selected Andrej Karpathy's reinforcejs[12] RL implementation. The agent's training parameters were set to the following:

```
spec.update = 'qlearn'; //algorithm
spec.gamma = 0.9; //discount factor
spec.epsilon = epsilon; //greedy policy
spec.alpha = 0.005; //learning rate
spec.experience_add_every = 5;
spec.experience_size = 10000;
spec.learning_steps_per_iteration = 5;
spec.num_hidden_units = 100 //n hidden units.
```

The only parameter that we changed for different throughputs, was the epsilon parameter that prescribes the exploration-exploitation probability. For the training phase, we set the epsilon value to 0.2. Afterwards, for the evaluation steps, we decreased the epsilon to 0.1 and in a second evaluation step to 0.0, in order to force the agent to apply only its trained experience. The agent was trained by the simulator for 10 minutes. After the 10 minutes, the evaluation of the trained model was conducted. Figure 2a depicts the collected reward during the training phase. The histogram shows that in more than 8000 decisions the agent gathered positive rewards and in comparison less than 2000 punishments and no reward. Already during the training the agent accomplished a good performance. Due to the exploration probability (0.2) the agent recommended also random activities. That explains why the agent has punishments and zero rewards. With the decreased epsilon value of 0.1, the agent achieves a better performance, while the zero values do no longer occur (see figure 2b). As we sat the epsilon parameter to 0.0, the agent has no longer punishments, since it only uses its trained model for making activity recommendations (see figure 2c). In order to show the distribution of the simulated training features, we generated a scatterplot that shows for an excerpt of features (i.e. ACR, eGFR, Hematuria, Potassium, Reward, Action) their distribution within the given ranges. All feature values are normalized in order to avoid scaling problems. It is easy to see in figure 2d that the features are broadly and equally distributed, except boolean values such as Hematuria. The feature Hematuria can be either zero, which means the evidence of *No Hematuria* or one which means the evidence of *Hematuria*. Moreover, the scatterplot indicates that 7 different actions were equally performed by the agent in order to get appropriate rewards that range from the values -1.0 to 1.0. The PoC has demonstrated that the agent improves its behaviour inside this virtual environment. With proceeding time, the agent gets on average more rewards than punishments. Considering the entire learning phase, the accumulated reward increases over time. By this PoC, we have shown that our proposed approach works as intended.

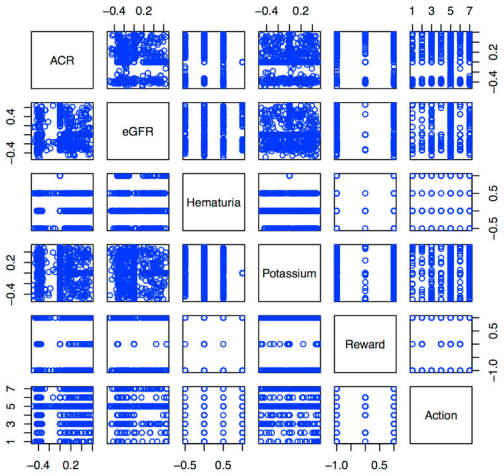
[12] https://cs.stanford.edu/people/karpathy/reinforcejs/



(a) Collected rewards with epsilon value 0.2

(b) Collected rewards with epsilon value 0.1

(c) Collected rewards with epsilon value 0.0



(d) Distribution of generated feature values

Fig. 2: Evaluation Results

6. Conclusion

In the presented work, we discussed how agents can be enabled to learn to perform by semantic task descriptions and the simulation of sensor events, the right actions in order to solve given tasks. Moreover, we show in our approach that combining semantic technologies as well as machine learning, improves the integration of new complex tasks and overcomes the cold-start problem because missing datasets that are required to train agents, can be generated by the presented simulator. Semantic technologies provide the agent with relevant knowledge about its environment and enable the agent to perform appropriate actions. Semantically represented guidelines that describe the characteristics of relevant observation features, facilitate the simulation of different tasks. Furthermore, domain experts are enabled to provide—in a simplified and formalized way—domain knowledge in order to program/train new agent instances via the presented simulator. Therefore, domain experts just require to create semantic task descriptions as well as a semantic representation of the agent's environment. The tasks can be arbitrary complex and the related observations can consist of high-dimensional sensor states, which can be handled by the agent via the DQN. The advantage of the once trained DQNs is, that they can be shared and reused by other agents in the environment. Moreover, before an agent acts in the real world, it is trained in a simulated and controlled environment. In this way, the agent can train

also without available real-world datasets for a task. The PoC demonstrates that an agent is able by our approach to master relatively fast a complex task such as the CKDPathway. As future work, we plan to conduct laboratory and field studies in order to evaluate the performance and generalisability of trained agents in comparison to rule-based agents and human experts. The extension and evaluation of the approach will be conducted within the scope of the European project *vCare*.

Acknowledgement

This work is supported by the European Commission in the context of the vCare project (No.769807).

References

[1] A.K.Mariappan, H.S.: A hybrid approach to solve cold start problem in recommender systems using association rules and clustering technique. International Journal of Computer Applications 74(4), 17 – 23 (July 2013)
[2] Carlson, A., et al.: Toward an architecture for never-ending language learning. In: AAAI. vol. 5, p. 3. Atlanta (2010)
[3] Di Noia, T., Ostuni, V.C.: Recommender Systems and Linked Open Data, pp. 88–113. Springer International Publishing, Cham (2015), https://doi.org/10.1007/978-3-319-21768-0_4
[4] Godoy, D., Amandi, A.: An agent-based recommender system to support collaborative web search based on shared user interests. In: Haake, J.M., Ochoa, S.F., Cechich, A. (eds.) Groupware: Design, Implementation, and Use. pp. 303–318. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
[5] Lika, B., Kolomvatsos, K., Hadjiefthymiades, S.: Facing the cold start problem in recommender systems. Expert Systems with Applications 41(4, Part 2), 2065 – 2073 (2014), http://www.sciencedirect.com/science/article/pii/S0957417413007240
[6] Mantrach, A.: Cold start solutions for recommender systems, https://research.yahoo.com/_c/uploads/SeminarUCSD.pdf
[7] Mehta, R., Rana, K.: A review on matrix factorization techniques in recommender systems. In: 2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA). pp. 269–274 (April 2017)
[8] Merkle, N., Kämpgen, B., Zander, S.: Self-service ambient intelligence using web of things technologies. In: Proceedings of the 1st Workshop on Semantic Web Technologies for Mobile and Pervasive Environments co-located with the 13th Extended Semantic Web Conference (ESWC 2016), Heraklion, Greece, May 29, 2016. pp. 1–10 (2016), http://ceur-ws.org/Vol-1588/paper3.pdf
[9] Merkle, N., Zander, S.: Improving the utilization of aal devices through semantic web technologies and web of things concepts. Procedia Computer Science 98(Supplement C), 290 – 297 (2016), http://www.sciencedirect.com/science/article/pii/S1877050916321779, the 7th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2016)/The 6th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2016)/Affiliated Workshops
[10] Middleton, S.E., Alani, H., Roure, D.D.: Exploiting synergy between ontologies and recommender systems. CoRR cs.LG/0204012 (2002), http://arxiv.org/abs/cs.LG/0204012
[11] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR abs/1312.5602 (2013), http://arxiv.org/abs/1312.5602
[12] Musto, C., et al.: Introducing linked open data in graph-based recommender systems. Information Processing and Management 53(2), 405 – 435 (2017), https://doi.org/10.1016/j.ipm.2016.12.003
[13] Nouali, O., Belloui, A.: Using semantic web to reduce the cold-start problems in recommendation systems. In: 2009 Second International Conference on the Applications of Digital Information and Web Technologies. pp. 525–530 (Aug 2009)
[14] Palau, J., et al.: Collaboration analysis in recommender systems using social networks. In: Cooperative Information Agents VIII. pp. 137–151. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
[15] Peis, E., del Castillo, J.M.M., Delgado-López, J.A.: Semantic recommender systems. analysis of the state of the topic. Hipertext.net 6, (online) (2008), http://www.hipertext.net/english/pag1031.htm
[16] Rubens, N., Elahi, M., Sugiyama, M., Kaplan, D.: Active Learning in Recommender Systems, pp. 809–846. Springer US, Boston, MA (2015), https://doi.org/10.1007/978-1-4899-7637-6_24
[17] Schein, A.I., Popescul, A., Unger, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: 25th Annual Internation ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 253–260. Tampere, Finland (2002)
[18] Settles, B.: Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison (2010), http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf
[19] Son, L.H.: Dealing with the new user cold-start problem in recommender systems: A comparative review. Information Systems 58, 87 – 104 (2016), http://www.sciencedirect.com/science/article/pii/S0306437914001525
[20] Thanh-Tai, H., Nguyen, H.H., Thai-Nghe, N.: A semantic approach in recommender systems. In: Future Data and Security Engineering. pp. 331–343. Springer International Publishing, Cham (2016)
[21] Tomeo, P., et al.: Exploiting linked open data in cold-start recommendations with positive-only feedback. In: Proc. of the 4th Spanish Conference on Information Retrieval. pp. 11:1–11:8. CERI '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2934732.2934745
[22] Yang, R., Hu, W., Qu, Y.: Using semantic technology to improve recommender systems based on slope one. In: CSWS. pp. 11–23. Springer (2012), http://dblp.uni-trier.de/db/conf/csemws/csws2012.html#YangHQ12