Model Driven Reinforcement Learning and Its Application

2022 Fall KAIST AI611 Final Project

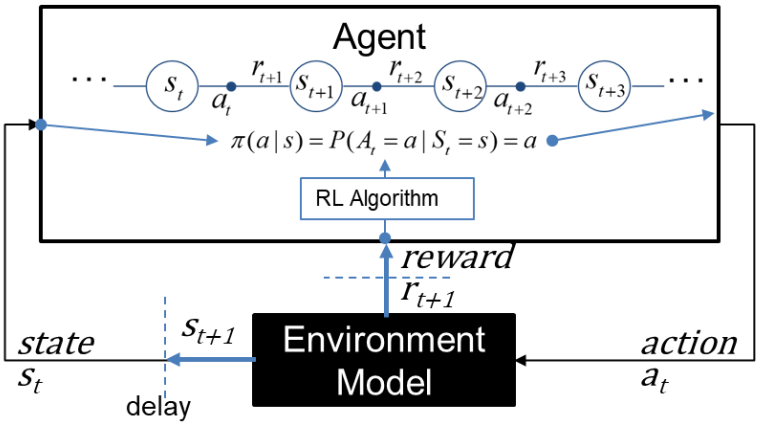
20175563 Kun-Chul Hwang

When applying the reinforcement learning upon the system engineering approach of modular and hierarchical system decomposition and integration method, the key challenges are to develop a training environment that meets the conceptual design specification of a complex physical system with components, and to conduct the integrated simulation of those components. To ease this modeling and simulation process, I have developed a novel model-driven reinforcement learning framework using the Discrete Event Simulation Formalism, so-call DEVS (Discrete Event System Specification). This final technical report introduces the overall development of the novel reinforcement learning framework, Model Driven Reinforcement Learning based on DEVS Software Architecture.

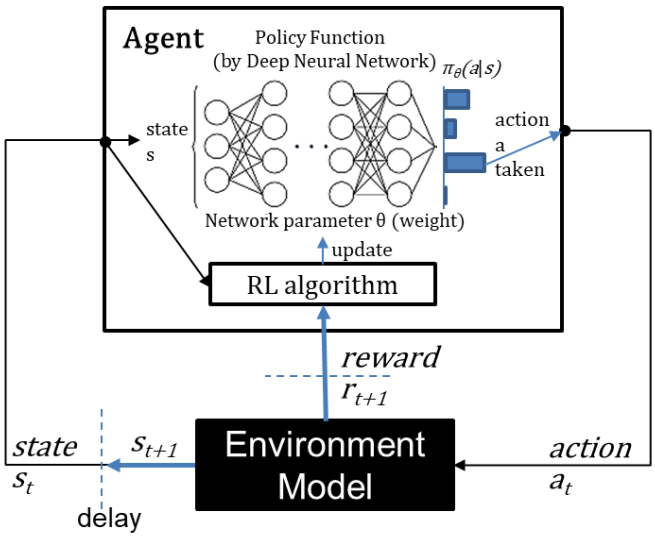
1

Deep Reinforcement Learning and its Challenge

We call it a control task that is a problem of making decision of an agent action or a controller command to a system or a environment input in order to get a required result. RL (Reinforcement Learning) is to resolve the control task in the way of machine learning to use the experience from the environment in make the proper action to it. Below figure show a general configuration of reinforcement learning. Here, the agent is modelled with the MDP (Markov Decision Process) that makes a decision, using only current state. And model-free means the agent of learning the proper action does not know about the environment model. DRL (Deep Reinforcement Learning) is to use DNN (Deep Neural Network) for the policy function in the RL.



Generalized Model free RL Configuration



Generalized DRL Configuration

Since DNN can work as a universal function approximator with universality of being applicable in wide range and excellency of resolving a complex problem, it can be effectively used as the policy function of the agent. Now, it becomes an essential technique with its outstanding performance in a wide range of application.

However, there are still the obstacles or intrinsic problems when applying that DRL into the real system development process. The first thing is that there is no custom RL environment for your application. (You need to build your RL environment simulation from the scratch). The second thing you may suffer from is that there is too large action space to be explore, or poor reward design so-called sparse reward. Simply, RL is a kind of try-and-error tuning process, and you do a bunch of action tries to the tuning process. If there are too large action space, you will suffer from finding the right actions.

About the Author: Kun-Chul Hwang is with Maritime Technology Research Institute of Agency for Defense Development, as a Principal Researcher, and doing his Ph.D. works of naval weapon systems modeling & simulation at KAIST Robotics.
(e-mail: hkchul@add.re.k / lionhrt@kaist.ac.kr)

The chart below shows an agent's RL performance (the reward score) with respect to the simulation iteration numbers. In this example, the DRL agent is not learning since the proper action is not made due to the large action space to be explored.

To tackle this intrinsic RL problems, this technical article introduces a novel RL method, so-called Model Driven Reinforcement.

The key ideas of the proposed method consist of 1) Making the custom RL environment model in a MDA (Model Driven Architecture), which is to develop a simulation SW (software) in a Lego-like component based software development approach. For example (below figure), we may develop a USV guidance algorithm based on RL (the collision avoidance controller (VO controller) of the USV). We may be forced to develop the custom environment and RL agent (VO controller) following the SW component design.

Thus, the figure below shows the idea that come from System Engineering Perspective of building up a custom RL environment, which is made in a modular and hierarchical way.

2) the second idea is to constrain action space with a given boundary so that the agent does not wander the action space but quickly find the proper action in the constraint space. Human has its own motto, just like "Honesty is the best policy", so that his action is usually selected with the constraint. This means there is a high-level doctrine or a boundary condition on human policy. With this intuition of human action, the proposed RL method incorporates a high level doctrine or a system operational information to the action space. With the operational information, the reinforcement learning efficiency is expected due to the reduced action space to be explored.

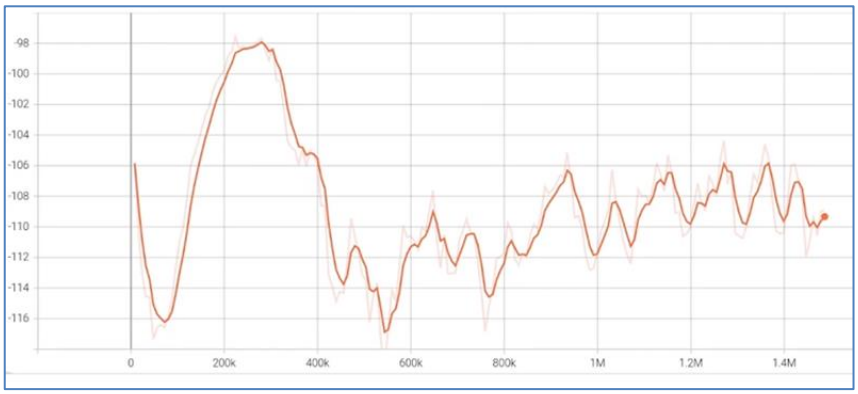
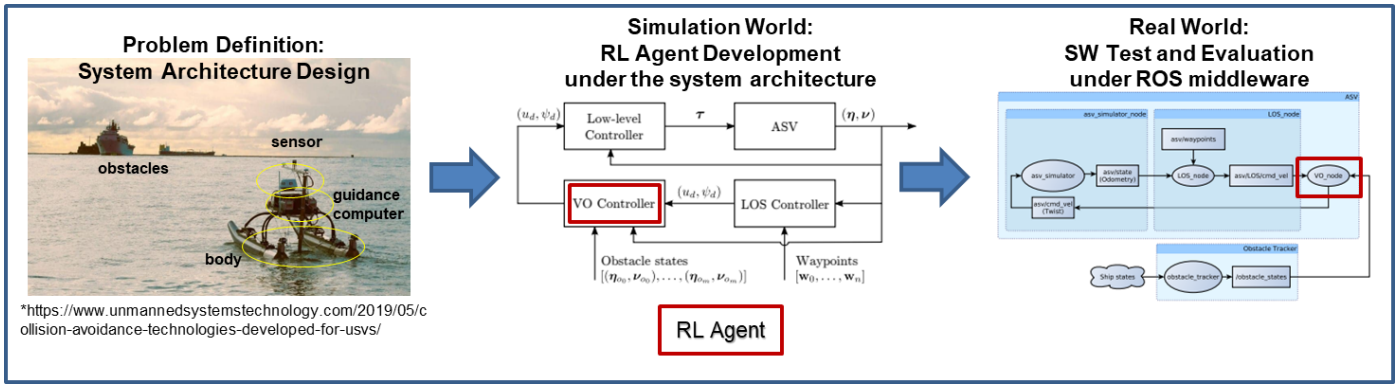


*image from ray/tune/episode_reward_mean (github.com/ray-project/ray/issues/24028)

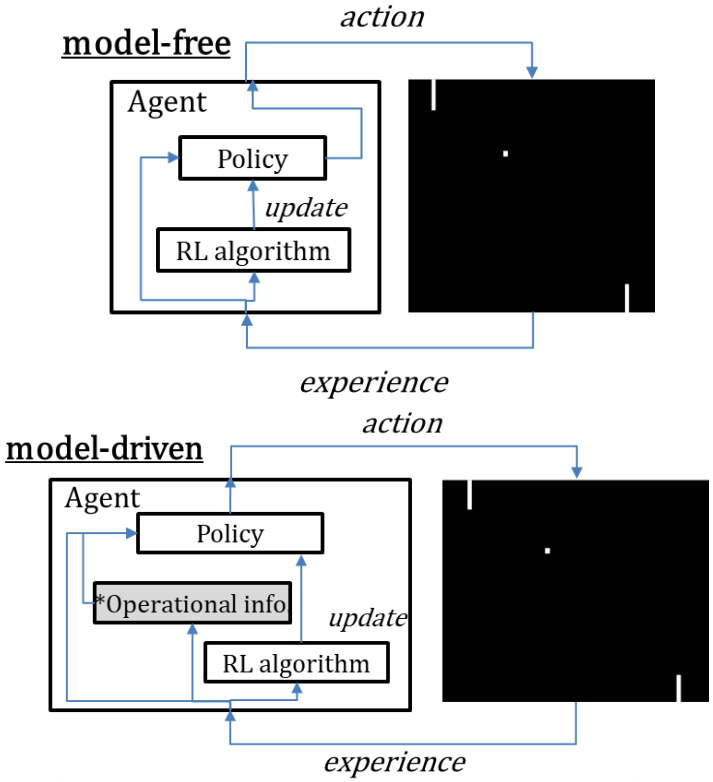


Model Driven Architecure of Lego-like Software Development Approach

2. Model Driven Reinforcement Learning

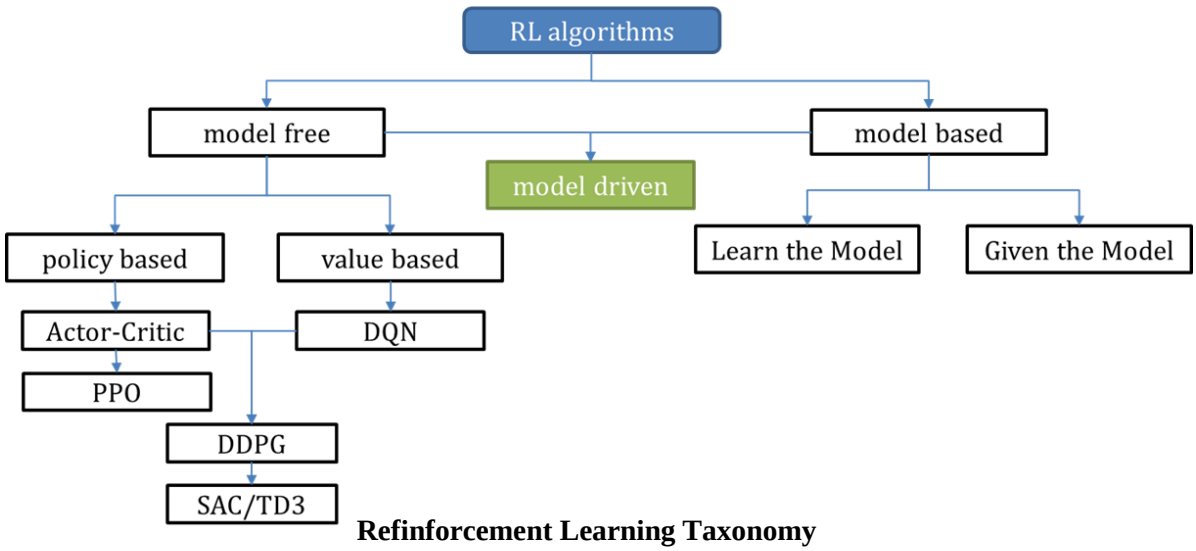
The below figure shows the taxonomy of RL. It categorizes the RL into model-free, model-based RL and the proposed model-driven RL. Model free RL approach has its advantage of not requiring the information about the environment model. Thus, it takes much time for the RL algorithm to converge the action in a certain level. Contrast to the model free, model-based gives the information about the environment. This model information serves as a guideline for the action selection, so that the method has the benefit of the sample efficiency about the action. However, model-based needs the work to build an environment model, that is hard or sometimes impossible. Model-driven approach proposed is the idea to mix those approaches, giving the model information in minimum, which is defined as the operational information to be consist of operation mode and action boundary

Let's see below comparison between model-free and model-driven RL application Ping-Pong Game. In model-free, experience is composed with state, reward information. However, model-driven give the extra information of the operation information composed with about the operation mode, and action boundary. Here. I assume the operation mode as RECEIVE and SEND. The action space is bounded with the modes



*Operational info. of the pong game : receive , send

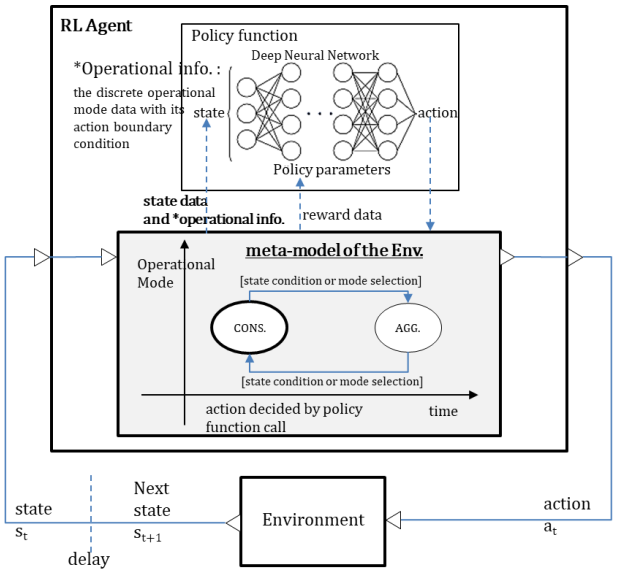
Model-free RL vs Model-driven RL



Refinforcement Learning Taxonomy

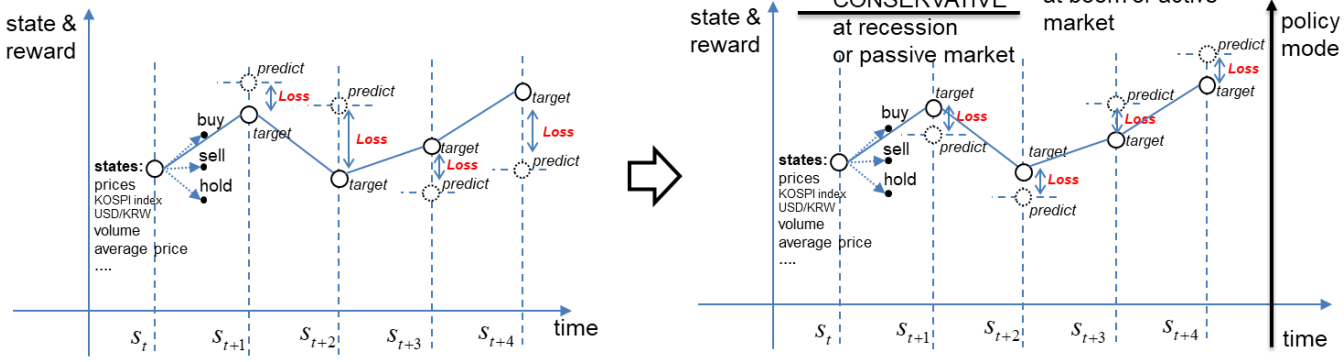
Then, **what is advantage of Model Driven RL?** To explain its merits, let's think about a stock trading agent. In the RL process, the trading agent try his action between (buy, sell, hold) with respect to the stock state. Although it predicts an action to maximize the reward (the total money), there is an anticipation error, that is loss, between target and prediction. The error could be severe if we do not consider a high-level condition of the stock market. For example, our prediction should be conservative when the market is at recession since trading momentum is low and the stock price easily falls. In contrast, the prediction could be overestimated in the boom market to maximize the profit. Thus, if we know the extra information of the environment operation, it could help our prediction accuracy giving a guideline for the agent action. As you see the right chart below, the policy mode can guide the policy action giving the bias on the action, so as to maximize the reward

To realize this idea, **the model-driven RL architecture is proposed.**



Model Driven RL Architecture

Model-driven RL consists of meta-model of the environment and policy (DNN policy function + RL algorithm to update the policy function). The agent meta-model gives the operation information, that is the minimum information about the environment. This extra information is fed to the policy input with the state, helping faster learning of the RL agent. The Meta-model is modelled by a finite state machine having discrete event states that is the operational mode. This operational mode can be the sub-task of the desired action, helping a precise action. With the operation mode, an action boundary could come helping faster action convergence.



Why the agent's prediction fails?
- there are hidden states with importance

If a high-level doctrine, an operational mode is given, what happens?

Model Driven RL agent (trading agent) 's Learning Process

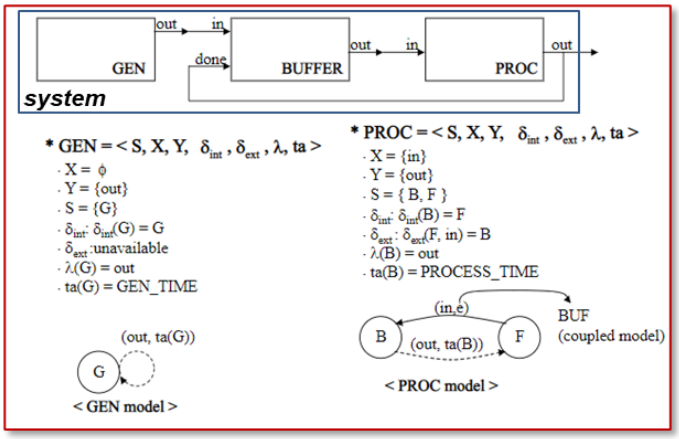
3. Related Works

- **Model based RL (or Hybrid RL)** needs to build up a model to generate an action trajectory for the agent to reference. But it is difficult to make the model [1)2)]
- **Hierarchical RL**: similar with the proposed idea. but lack of a concrete formalism to represent the meta-controller. and there is a complexity of making the controllers (inner and outter) to interact with the DNN [3)]
- **DEVS (Discrete EVent System Specification)** based framework for RL: mainly focusing on constructing the RL environment in a discrete event system modeling methodology. DEVS formalism is a kind of an extended finite state machine automata to represent the discrete event system in a more expressive way[4)]. But the proposed idea uses DEVS for generating the environment's operational information and use it for the RL training process

4. DEVS formalism and Implementation

- DEVS formalism give the underlying specification of the Model Driven Architecture for developing software. It is a meta-modeling formalism with features of the object-oriented, modular, and hierarchical system modeling and simulation methodology. A Lego-like component modeling method using the discrete event state automata (an extended finite state machine).

DEVS represents a system (or a simulation environment) with a coupled model (a structural model) and an atomic model (a behavioral model)



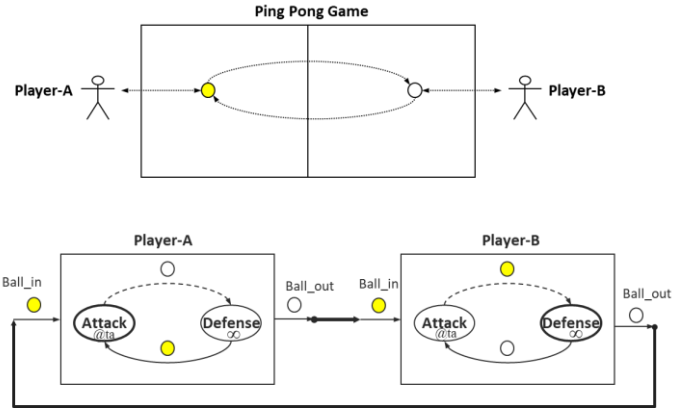
*Marlyn calfee, "Review DEVS Formalism Discrete-Event formalism ", lecture note

In above DEVS modeling example, the system model is represented by the coupled model which is decomposed with inner components. And the component's behavior is represented by the atomic model, not decompose any more. Thus, the coupled model (system) decomposed with GEN/BUFFER/PROC in their input/output port coupling. The atomic model (GEN/PROC) is represented by the extended finite state machine with the external transition (solid line) and internal transition (dotted line)

The external transition is triggered by input of the atomic model. And the internal transition triggered by dwelling time in a discrete state, that is ta(S)

When the internal transition made, an output is propagated into output port.

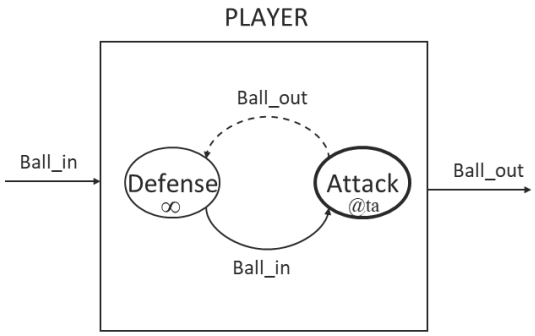
To clearly understand DEVS modeling and simulation, let's see the below DEVS ping-pong game



DEVS ping-pong game

In DEVS ping-pong game, a player is represented by the atomic model with the set element that is implemented as the programming interface (a class interface). The atomic model has input(X), output(Y), discrete states(S), external transition function(δ_{ext}) to define the discrete state from one to another in receiving the input, internal transition function(δ_{int}) to define the discrete state change in propagating output with a simulation time elapsed, time advanced function(ta). The atomic model's graphical notation is described as the circle is the discrete states (bold line circle means the initial state), the letter inside circle means the discrete state names. The @*num* means time advanced function to define the dwelling time *num.* in a discrete state. (∞ means the infinite dwelling time). The solid/dotted line means the external/internal transition function to direct the state transition from one to another.

The atomic DEVS model can be implemented in an object-oriented language with the class interfaces for each element of the DEVS model. Here, I have implemented the class in MATLAB



PLAYER = < X, Y, S, δ_{ext}, δ_{int}, ta, λ >

X = {Ball_in}, Y = {Ball_out}, S = {Defense, **Attack**} *initial state*

δ_{ext} (Defense, Ball_in)= Attack

ta (Attack) = ATTACK_TIME

δ_{int} (Attack, ATTACK_TIME) = Defense

λ (Attack, ATTACK_TIME) = Ball_out

DEVS atomic model set and notation

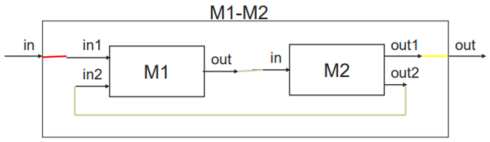
Atomic DEVS Model Class interface

Atomic DEVS	class CAtomic
X	bool AddInPorts(int num, ...);
Y	bool AddOutPorts(int num, ...);
S	Member Variable
δ_{ext}	bool ExtTransFn(const CMessage &)
δ_{int}	bool IntTransFn()
λ	bool OutputFn(CMessage &)
ta	TimeType TimeAdvanceFn()

```
classdef Player < CModel
   properties (Constant) %phase set
       WAIT=0;
       SEND=1;
   end

   properties
       m_phase % the discrete state variable (phase variable)
       condition %the key performance parameter of the player
   end

   methods
       function obj=Player(strName)
           if nargin>0
               obj.SetName(strName);
           end
           obj.condition = rand;

           obj.AddInports(2,'Receive','Serve');
           obj.AddOutports(2,'Send','Fail');

           %declare and initialize the discrete state
           obj.SetPhase('m_phase',obj.WAIT)
       end
       function ExtTransFn(obj, t, inport, data)
           if obj.m_phase==obj.WAIT
               if strcmp(inport,'Receive')||strcmp(inport,'Serve')
                   obj.m_phase=obj.SEND;
               else
                   error('Unexpected input in ExtTransFn!!!')
               end
           end
       end
       function OutputFn(obj,t)
           if obj.m_phase==obj.SEND

           elseif obj.m_phase==obj.WAIT

           else
               error('unexpected state in OutputFn !!!')
           end
       end

       function IntTransFn(obj,t)
           if obj.m_phase==obj.SEND
               if obj.condition > rand
                   obj.SetEvent('Send',[]);
               else
                   obj.SetEvent('Fail',[]);
               end

               obj.m_phase=obj.WAIT;
           else
               error('unexpectied state in IntTransFn!!!')
           end
       end


       function ta=TimeAdvanceFn(obj)
           if obj.m_phase==obj.SEND
               % N random numbers in the interval (a,b) with the
formula r = a + (b-a).*rand(N,1).
               ta=0.1+1.1*rand;
           else
               ta=Inf;
           end
       end
   end
end
```

A DEVS atomic model implementation (MATLAB)

The coupled DEVS model gives the structure of the system, which is to define the coupling relationship of the inner components. Thus, the coupled model's elements set is the input(X), output(x), coupling relationships (EIC:External-to-Input Coupling, EOC : External-to-Output coupling, IC : Internal Coupling), component's execution priority(SELECT)



CM = < X,Y,M,EIC, EOC, IC, SELECT>

X : input event set ;

Y : output event set ;

M : set of all component models in DEVS ;

EIC $\subseteq$ DN.IN $\times$ M.IN : external input coupling relation ;

EOC $\subseteq$ M.OUT $\times$ DN.OUT : external output coupling relation ;

IC $\subseteq$ M.OUT $\times$ M.IN : internal coupling relation ;

SELECT : $2^M - \varnothing \rightarrow$ M : tie-breaking selector.(M1,M2 priority)

DEVS coupled model set and notation

Coupled DEVS Model Class interface

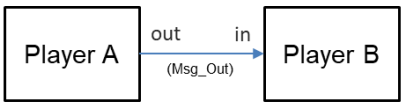
Coupled DEVS	class CCoupled
X	bool AddInPorts(int num, ...);
Y	bool AddOutPorts(int num, ...);
M	bool AddComponent(int num, ...);
EIC,EOC,IC	bool AddCoupling(CModel *src_model,const char *src_port, CModel *dst_model, const char *dst_port);
SELECT	bool SetPriority(int num, ...);

```
classdef PingPong < Cmodel
   properties
   end

   methods
       function obj=PingPong

           obj.SetName('Top Level Coupled Model');

           player_a=Player('a');
           player_b=Player('b');
           ref = Referee('referee'); % a referee added to the model

           obj.AddComponents(3,ref,player_a,player_b);
           obj.AddInports(1,'in');
           obj.AddOutports(1,'out');

           %EIC:External Input Coupling
           obj.AddCoupling(obj,'in',ref,'Reset');

           %IC:Internal Coupling
           obj.AddCoupling(player_a,'Send',player_b,'Receive');
           obj.AddCoupling(player_a,'Fail',ref,'Fail_a');
           obj.AddCoupling(player_b,'Send',player_a,'Receive');
           obj.AddCoupling(player_b,'Fail',ref,'Fail_b');
           obj.AddCoupling(ref,'Serve_a',player_a,'Serve');
           obj.AddCoupling(ref,'Serve_b',player_b,'Serve');

           %EOC:External Output Coupling
           obj.AddCoupling(ref,'Report',obj,'out'); %EOC
       end
   end
end
```

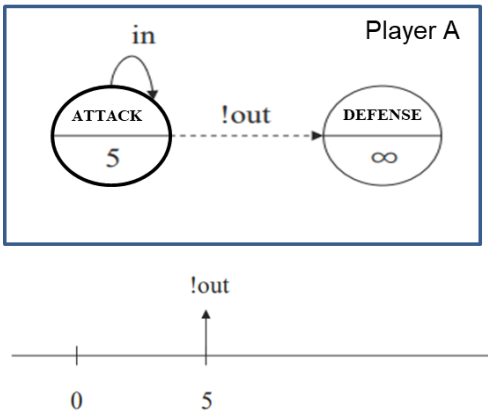
A DEVS coupled model implementation (MATLAB)

Once the DEVS models is constructed for a discrete event system. The model is executed in the follow sequence.

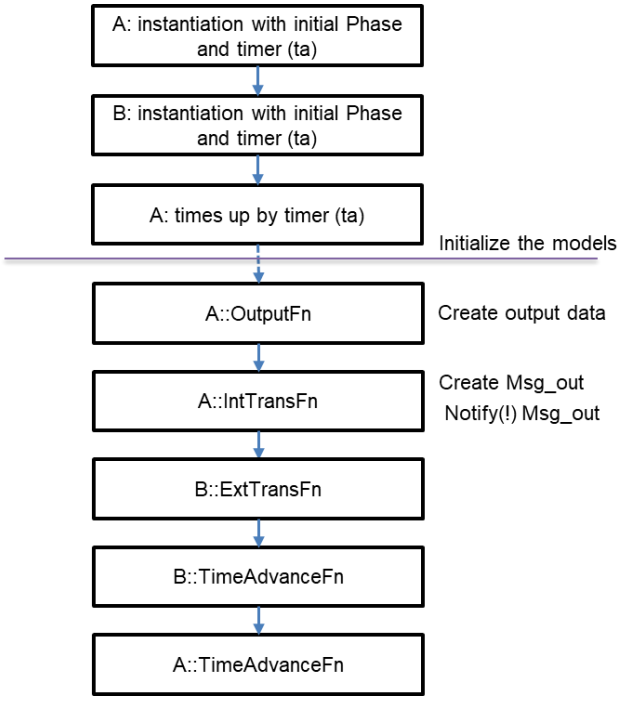
- coupled model instantiated with its atomic models instantiated. Each atomic model's scheduled time(next time trigger defined in time advanced function)
- once an atomic model is triggered with the simulation time elapsed, it generates an output and notify (!) the output event with the internal transition (dotted line) made.
- the output is propagated to a coupled model via the connected port. It now serves as the input event to trigger the external transition.
- Once the external transition made, the discrete state of the external transition model is changed, and the new state's time schedule is updated by calling the model's time advanced function.
- And the internal transition model's discrete state update made and finally, the model's time advanced function is called, finishing the function call order



User Defined Msg_Out transfer via out/in event port



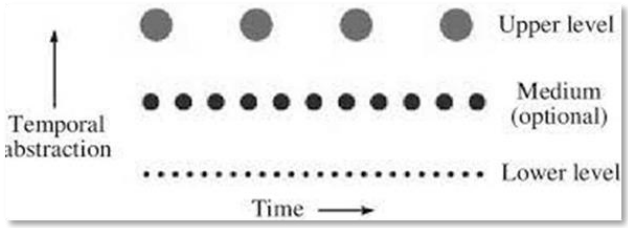
DEVS ping-pong game simulation time schedule



DEVS ping-pong game function call order

While OpenAI usually models an RL environment in a single class, DEVS can models that environment in a modular and hierarchical way that is more systematic approach.

The timing issue can be interpreted as the temporal abstraction for Semi-MDP (Markov Decision Process). In other words, DEVS modeling can implement the SMDP easily.



*Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence, 112(1):181–211.

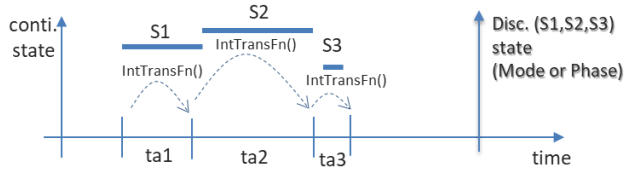
5. Model Driven RL using DEVS formalism

With this DEVS modeling and simulation formalism, it is possible to represent the RL environment in a more expressive way. For example, DEVS can control the execution timing in a time varying steps compared to the legacy RL environment framework, the OpenAI class interface,

Moreover, the timing issue, this DEVS approach can give the operational information, so-called operational mode or phase. This operational information can be used for the subtasks to define a precise action.

Atomic DEVS Model Class interface

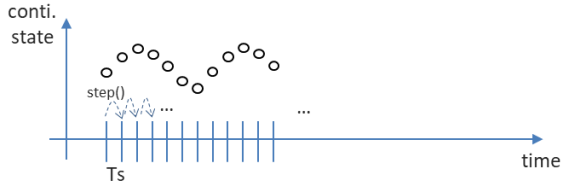
Atomic DEVS	class CAtomic
X	bool AddInPorts(int num, ...);
Y	bool AddOutPorts(int num, ...);
S	Member Variable
δ_{ext}	bool ExtTransFn(const CMessage &)
δ_{int}	bool IntTransFn()
λ	bool OutputFn(CMessage &)
ta	TimeType TimeAdvanceFn()



VS

OpenAI Environment Class interface

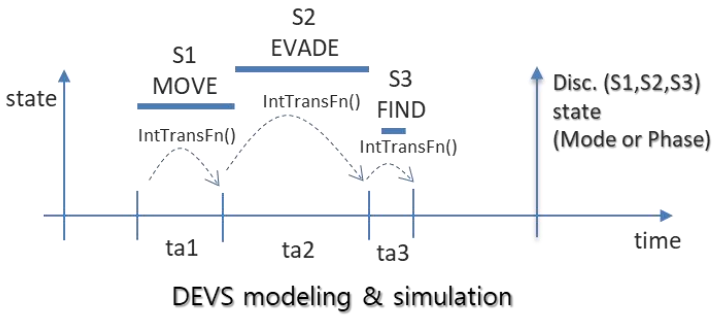
Constructor
step
reset
render



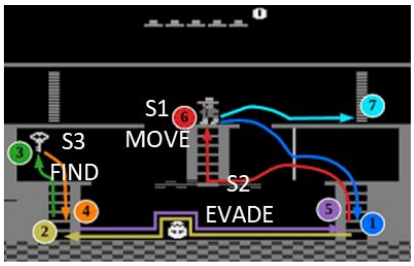
Model Driven RL / OpenAI RL class interfaces and simulation timing

Another advantage to use the DEVS RL framework, that is model-driven RL, is its capability to represent the environment model in a system engineering perspective. While OpenAI usually models an RL environment in a single consolidated class, DEVS can models that environment in a modular and hierarchical way that is more systematic approach.

When applying the reinforcement learning upon the system engineering approach of modular and hierarchical system decomposition and integration method, the key challenges are to develop a training environment that meets the conceptual design specification of a complex physical system with components, and to conduct the integrated simulation of those components. With model-driven RL framework, we can easily do the modeling and simulation process. For example, If the conceptual models can be decomposed and each component is expressed in detail, the proposed framework can make the modeling and simulation works easier.

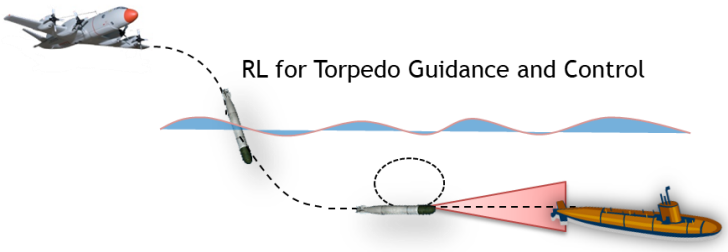


DEVS modeling & simulation



D. Lyu etal, Reinforcement Learning on Hierarchical Knowledge for Transferable and Adaptive Decision Making, 2019, AAAI

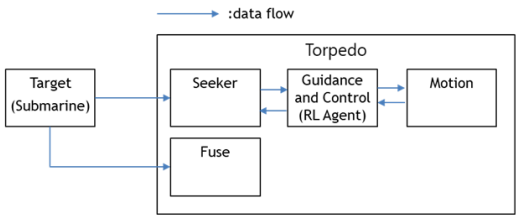
Model Driven RL's operational mode for the agent's subtasking

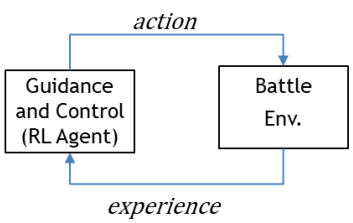


DEVS Model Class Structure
(Model Driven RL Architecture)

VS

OpenAI Environment Class Structure





Model Driven RL's environment model configuration
(A torpedo guidance and control development example)

6. How it works - Model Driven RL

To use DEVS modeling environment in RL as model-driven RL, the modification was made to the classical DEVS interfaces.

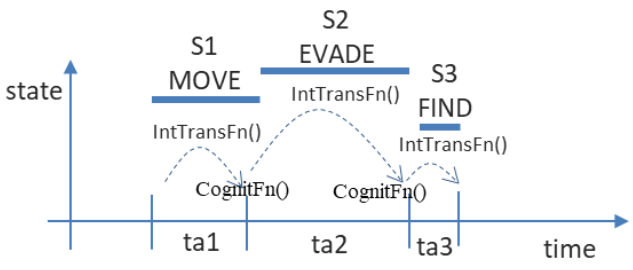
I have added the extra function, so-called CognitFn( ), to the conventional Atomic DEVS model interface. CognitFn( ) is called after one step the agent simulation done. CognitFn() is a kind of RL information generation function given the state observation . This is to mimic the human cognition process which is followed every time step in our brain after perception of sensing something is made.

Thus, the model driven RL's function execution sequence is illustrated as below.

FSM modeling Interface

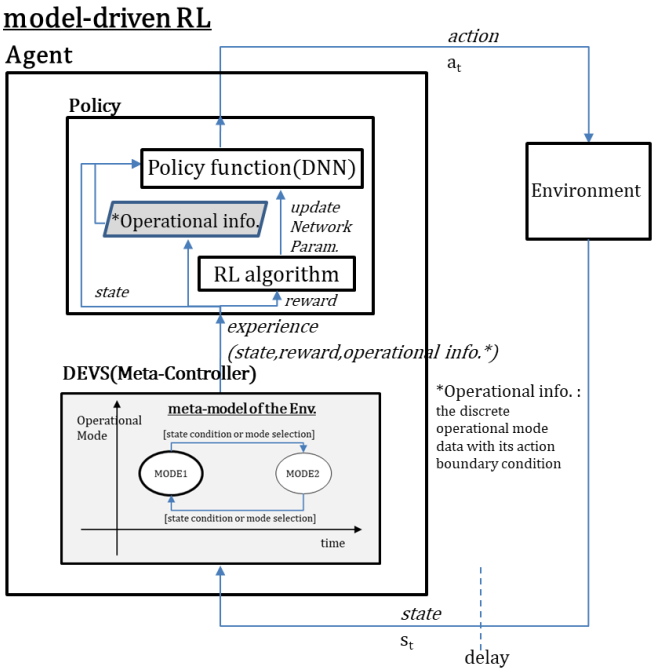
Atomic DEVS	class CAtomic
X	bool AddInPorts(int num, ...);
Y	bool AddOutPorts(int num, ...);
S	Member Variable
δ_{ext}	bool ExtTransFn(const CMessage &)
δ_{int}	bool IntTransFn()
λ	bool OutputFn(CMessage &)
ta	TimeType TimeAdvanceFn()
C	state, reward, done, mode = CognitFn()

*Here, mode of CognitFn( ) is the operational info. of the agent structured with mode name and action space boundary.
*Once the IntTransFn() is called and the model update is finished, CognitFn() is called for reinforcement learning

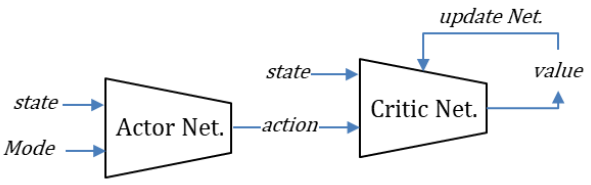


Model Driven RL's CognitFn( ) and Its function call order

Once CognitFn() generates the experience (state, reward, done, operational info.), the policy function have to process the input and updates the network parameters. To do this, the below shows the DNN architecture of the proposed framework.



Policy function(DNN)



RL algorithm

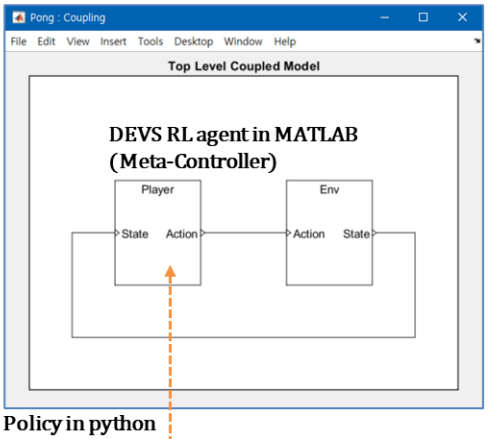
A Modified DDPG algorithm to incorporate the operational info.(Mode) and action boundary

Model Driven RL network architecture

7. Model Driven RL application

To test the feasibility of the proposed RL, I have implemented a simple ping-pong game (pong game). It is composed of a player (bar) and the wall (the environment)

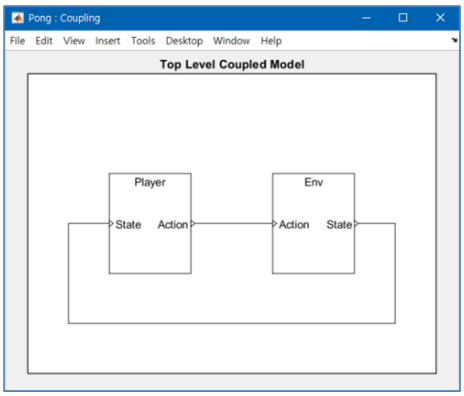
The agent and the env. model is implemented in MATLAB and the policy is implemented in PYTHON.

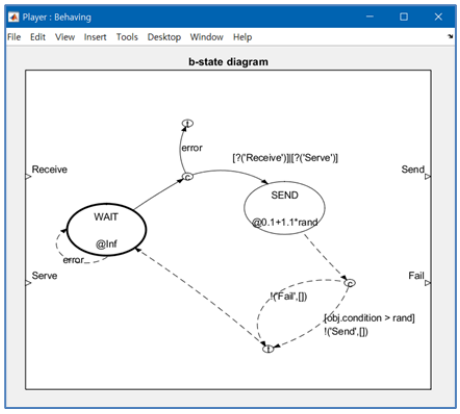


Policy in python

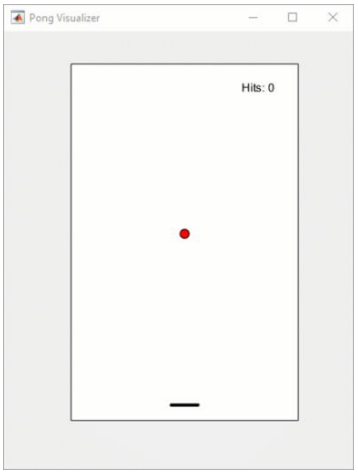
```
def main():
    env = gym.make('Pong-v0')
    memory = ReplayBuffer()

    q, q_target = QNet(), QNet()
    q_target.load_state_dict(q.state_dict())
    mu, mu_target = MuNet(), MuNet()
    mu_target.load_state_dict(mu.state_dict())

    score = 0.0
    print_interval = 20

    mu_optimizer = optim.Adam(mu.parameters(), lr=lr_mu)
    q_optimizer  = optim.Adam(q.parameters(), lr=lr_q)
    ou_noise = OrnsteinUhlenbeckNoise(mu=np.zeros(1))

    reward_list = []
    for n_epi in range(10000):
    #for n_epi in range(100):
        s = env.reset()
        done = False
    . . .
```

Model Driven RL Policy in PYTHON

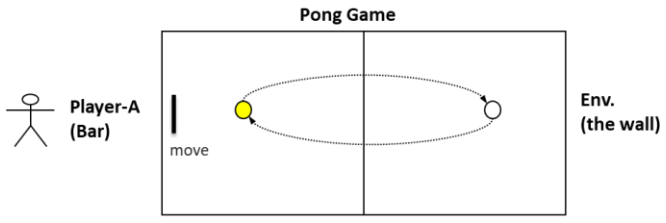


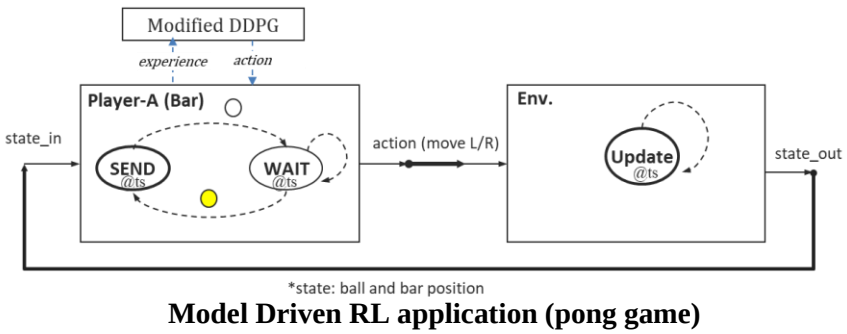


Model Driven RL enviroment and the agent implemention in MATLAB



Initial test run of pong game





*state: ball and bar position

Model Driven RL application (pong game)

8. Conclusion and Discussion

To apply RL into the system engineering process, a noble RL approach of model-driven RL is proposed.

- the proposed RL approach is to combine the system specification formalism (DEVS) with RL. Thus, it make the RL simulation environment in Model Driven Architecture.
- the proposed RL is to extend the experience data fed to the agent from the environment with the operational information that is represented by the agent's DEVS model. The operational information is intended for the action boundary helping the learning speed faster.
- model-driven RL framework is developed to make it easier the RL simulation to realization issue, incorporating a model-driven architecture, DEVS formalism.
- Model-driven RL example is initially tested in this course. But the detailed analysis, the performance comparison with the legacy RL method and the advantage proof of the proposed method, is not followed due to time limit of submission. It needs more time of implementation and testing. Future work will cover this analysis.

With this course and work, I have got the idea of a novel RL framework and had great discussion on the RL research topics.

- Thank you

REFERENCE

[1] Moerland, Thomas M., Joost Broekens, and Catholijn M. Jonker. "Model-based reinforcement learning: A survey." arXiv preprint arXiv:2006.16712 (2020).

[2]Tesauro, Gerald, et al. "A hybrid reinforcement learning approach to autonomic resource allocation." 2006 IEEE International Conference on Autonomic Computing. IEEE, 2006.

[3] Nachum, Ofir, et al. "Data-efficient hierarchical reinforcement learning." Advances in neural information processing systems 31 (2018).

[4] David, Istvan, and Eugene Syriani. "Devs Model Construction As A Reinforcement Learning Problem." *2022 Annual Modeling and Simulation Conference (ANNSIM)*.