Controlling Intelligent Agents in Virtual World

The OpenCog Embodiment Component

Nil Geisweiller

Novamente LLC

Xiamen University
AGI Summer School 2009

Outline

What is Opencog Embodiment Component?

Embodiment component (formerly called PetBrain)

Interface OpenCog for virtual world agents or robots

Embodiment Component Function

Goals:

- Environment where to teach and test OpenCog

Embodiment Component Function

Goals:

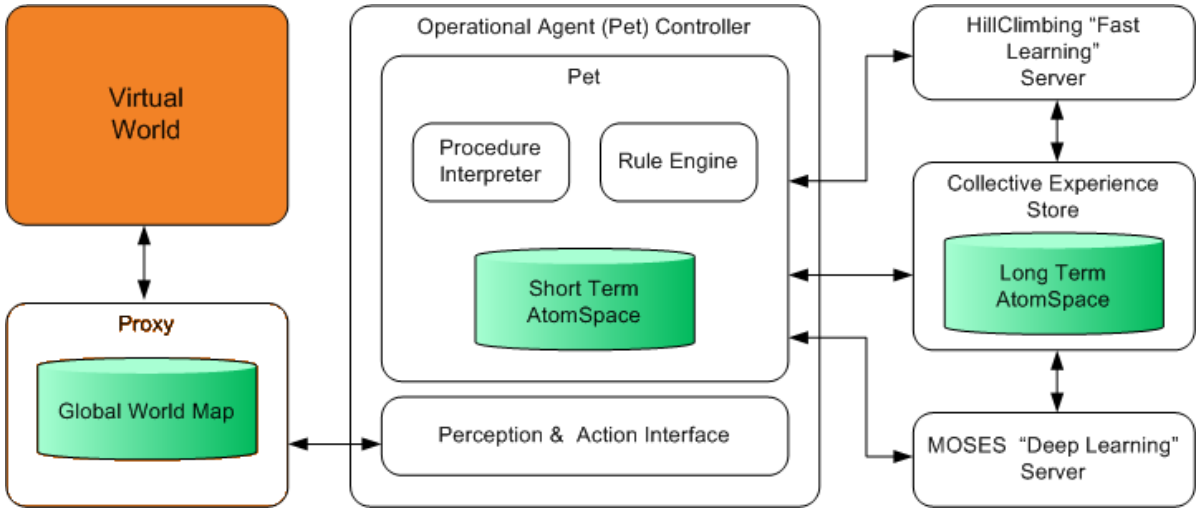
- Environment where to teach and test OpenCog
- Virtual World $\Rightarrow$ Easier than real world

Embodiment Component Function

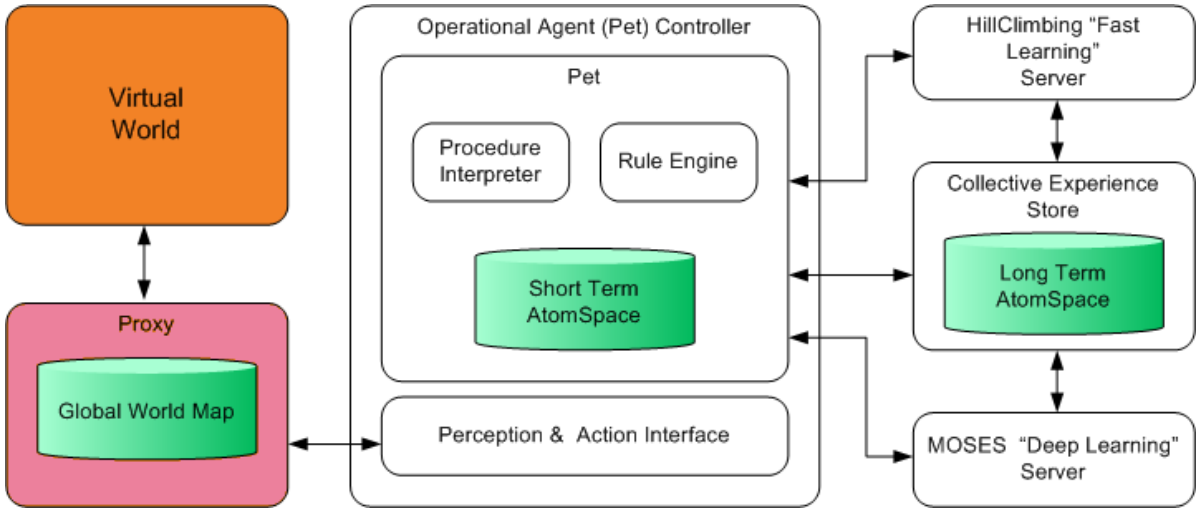
Goals:

- Environment where to teach and test OpenCog
- Virtual World $\Rightarrow$ Easier than real world
- But robot interface as well (in development)

Embodiment Component Overview

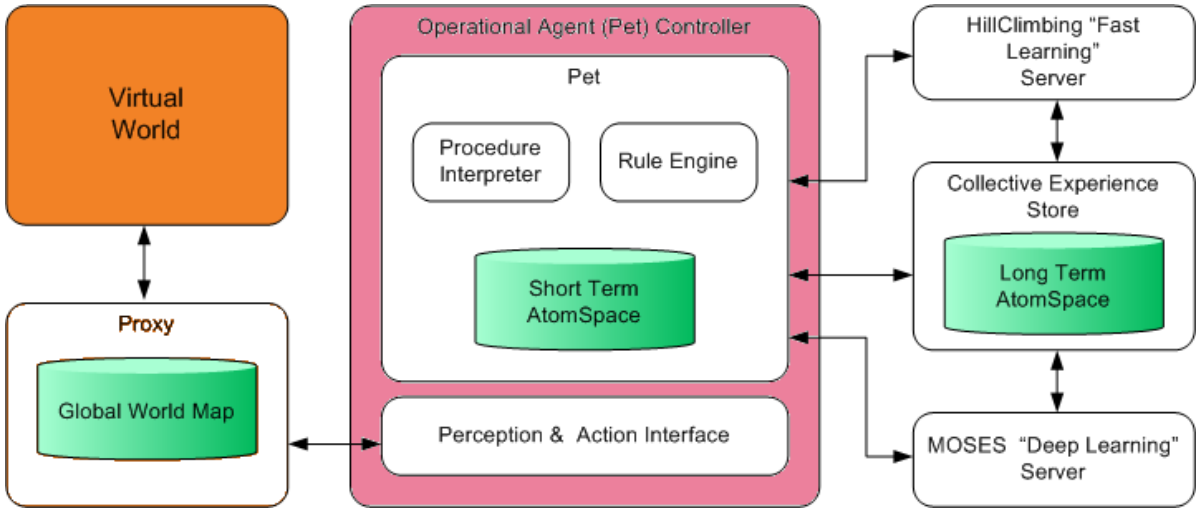


Embodiment Component Overview



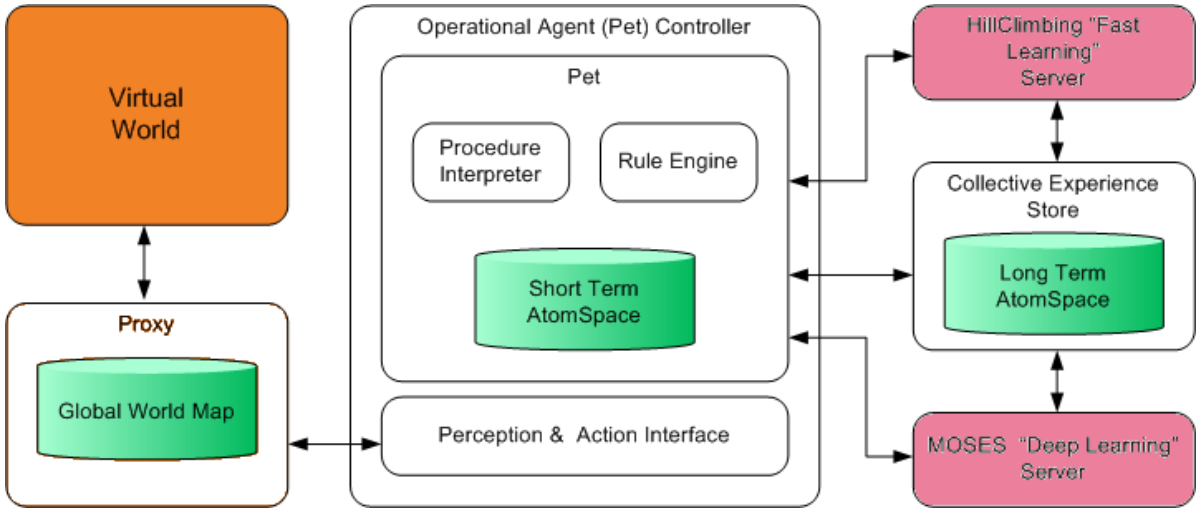
1. Proxy, interface between the virtual world and Embodiment

Embodiment Component Overview



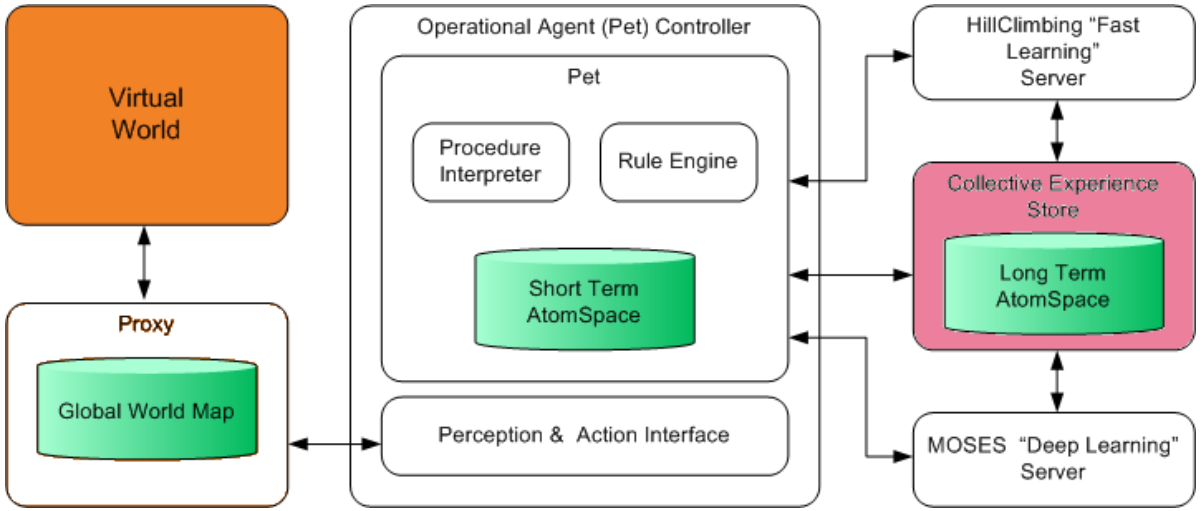
1. Proxy, interface between the virtual world and Embodiment
2. Operational Agent (Pet) Controller (OPC), reactive brain

Embodiment Component Overview



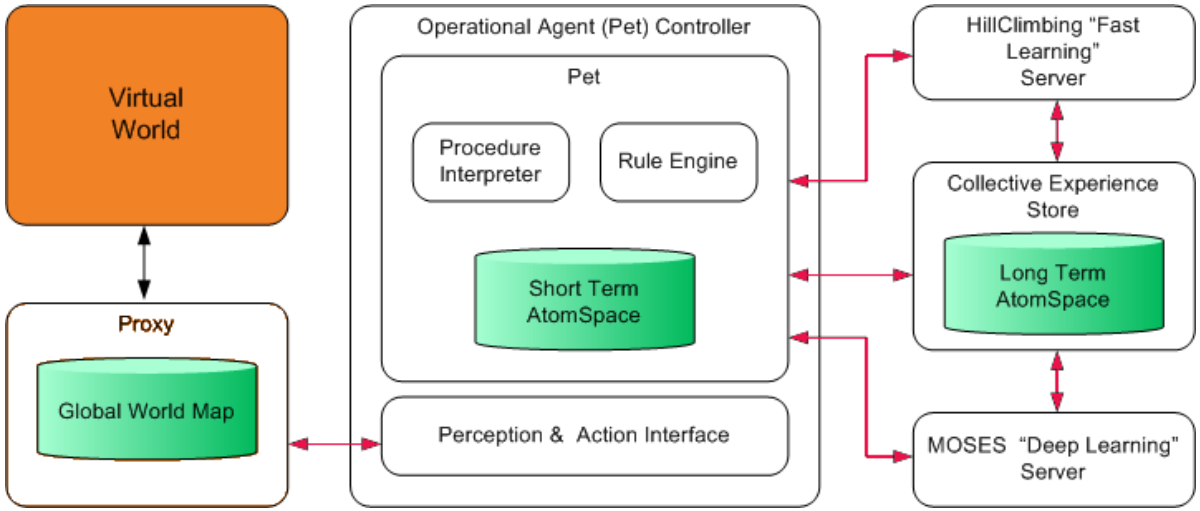
1. Proxy, interface between the virtual world and Embodiment
2. Operational Agent (Pet) Controller (OPC), reactive brain
3. Learning Server (LS), imitation learning for now

Embodiment Component Overview



1. Proxy, interface between the virtual world and Embodiment
2. Operational Agent (Pet) Controller (OPC), reactive brain
3. Learning Server (LS), imitation learning for now
4. Collective Experience Store (CES) (not implemented yet)

Embodiment Component Overview

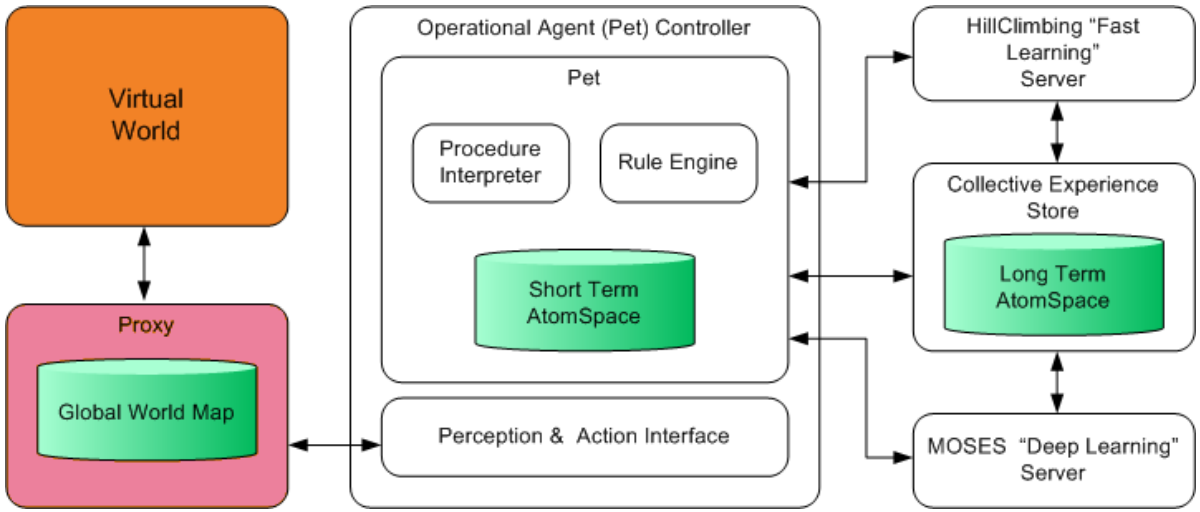


1. Proxy, interface between the virtual world and Embodiment
2. Operational Agent (Pet) Controller (OPC), reactive brain
3. Learning Server (LS), imitation learning for now
4. Collective Experience Store (CES) (not implemented yet)
5. Router: communication with Sockets (distributed or not)

Outline

Outline

Embodiment Proxy



Proxy main function

Abstract away all details of interacting with a Virtual World

For the moment Embodiment Proxy for

- Multiverse (implemented)

For the moment Embodiment Proxy for

- Multiverse (implemented)

- realXtend (almost complete)

For the moment Embodiment Proxy for

- Multiverse (implemented)

- realXtend (almost complete)

- Nao Robot (in development)

Embodiment Proxy sub-functions

Proxy handles:

- Input stream of perceptual data

Embodiment Proxy sub-functions

Proxy handles:

- Input stream of perceptual data
- Output stream of actions and action feedback (`failure` or `success`)

Embodiment Proxy sub-functions

Proxy handles:

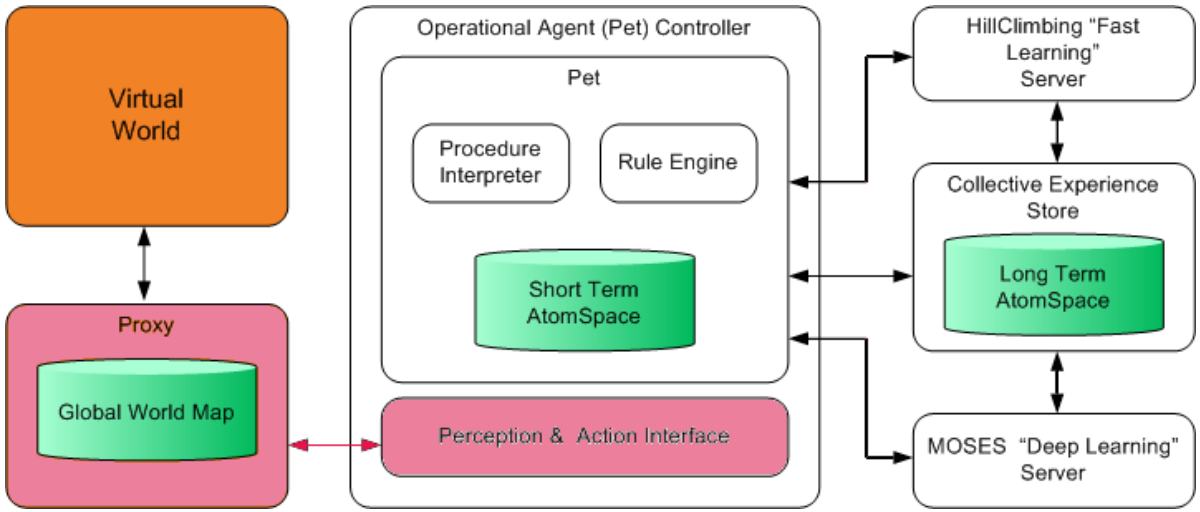
- Input stream of perceptual data
- Output stream of actions and action feedback (`failure` or `success`)
- Maintaining a Global World Map (GWM)
  - Shared GWM for all agents
    $\Rightarrow$ reduce message traffic agents $\leftrightarrow$ virtual world

Embodiment Proxy sub-functions

Proxy handles:

- Input stream of perceptual data
- Output stream of actions and action feedback (`failure` or `success`)
- Maintaining a Global World Map (GWM)
  - Shared GWM for all agents
    $\Rightarrow$ reduce message traffic agents $\leftrightarrow$ virtual world
- Commands from a human avatar
  - messages to the agents, order, queries
  - meta-commands (load agent, etc)

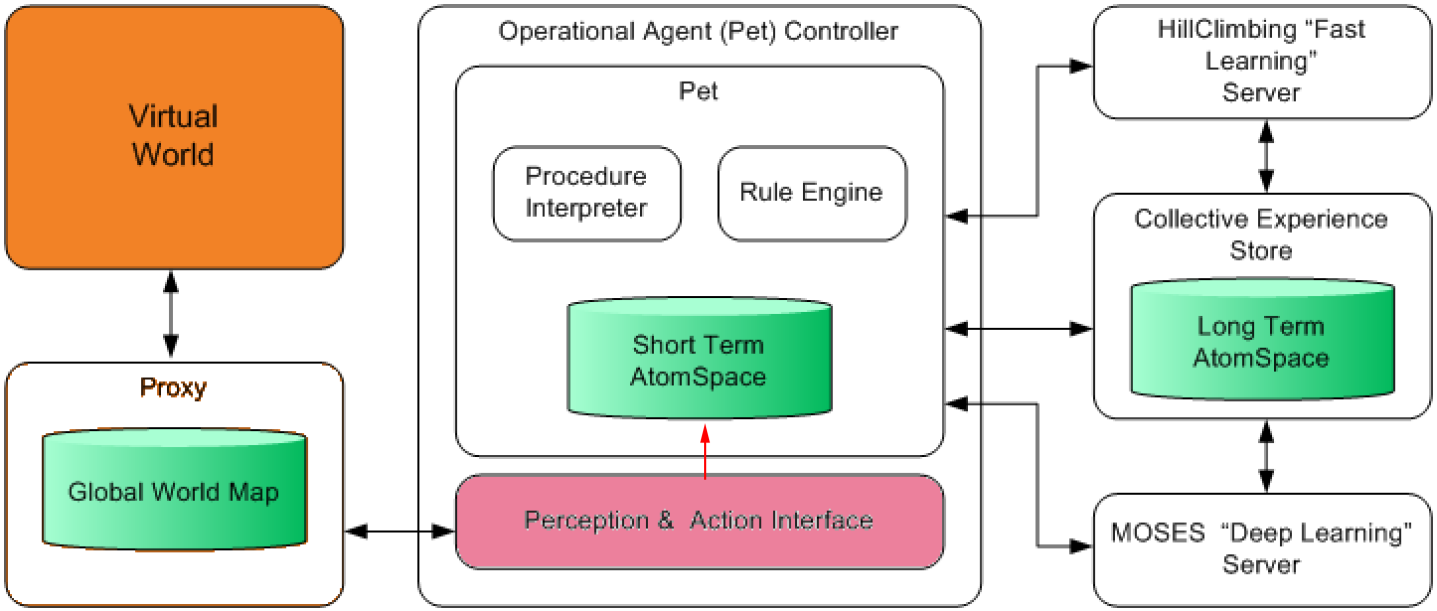
Embodiment Proxy ↔ PAI



Proxy communicates with the OPC by the intermediate of the Perception Action Interface.

Outline

Perception Action Interface Function



- Convert Perceptions coming from the Proxy into atoms in the agent's atomSpace

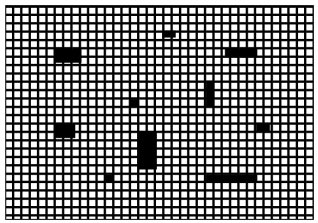
Perception Action Interface, perception → atoms

$\Rightarrow$ Proxy $\Rightarrow$ PAI $\Rightarrow$

```
AtTimeLink
    TimeNode: "15:32:24.182"
    EvaluationLink
        PredicateNode: "actionDone"
        ListLink
            AvatarNode: "owner"
            Node: "grab"
            ListLink
                ObjectNode: "ball_id"
```

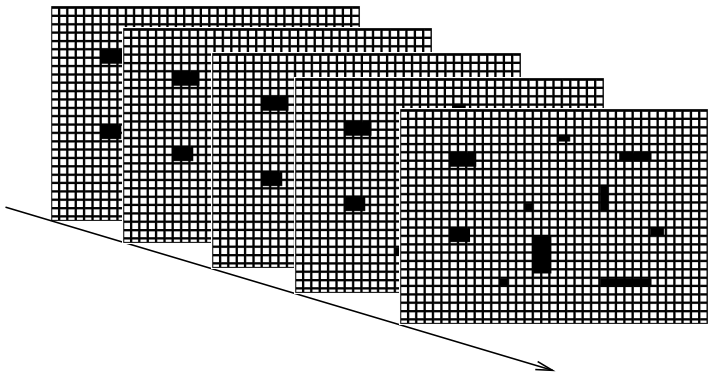
1. Pet watches the avatar grabbing a ball
2. Proxy sends perception to PAI
3. PAI writes the perception into the Pet's atomSpace

Embodiment AtomSpace Extensions, Space and Time servers



- Space Server: SpaceMap wrapped in atom, associative container for fast retreival.
  - SpaceMap:
    - 2D Grid
    - Objects inside
    - Spacial functions and predicates, `distance`, `near`, ...

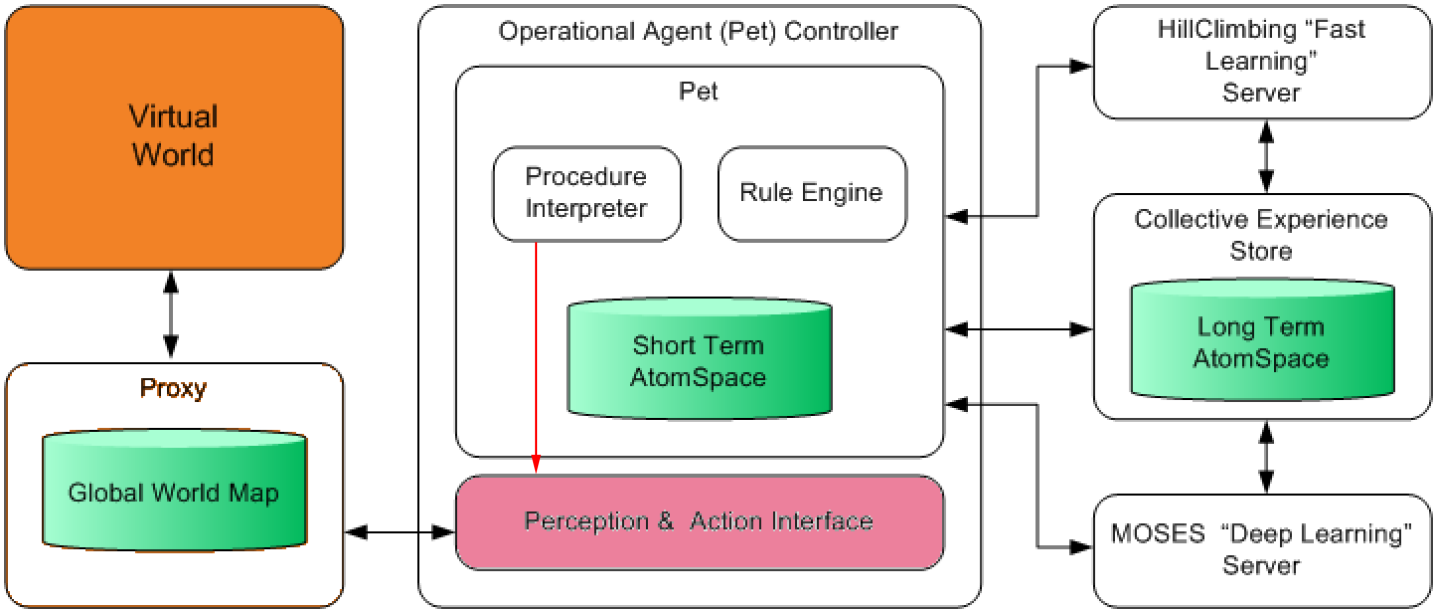
Embodiment AtomSpace Extensions, Space and Time servers



- Space Server: SpaceMap wrapped in atom, associative container for fast retreival.
  - SpaceMap:
    - 2D Grid
    - Objects inside
    - Spacial functions and predicates, `distance`, `near`, ...
- Time Server: indexing timestamped atoms for fast retrieval

Episodic memory

Combo interpreter actions to action plan



- Convert actions coming from the Combo interpreter into action plans (sequence of actions) to the Proxy

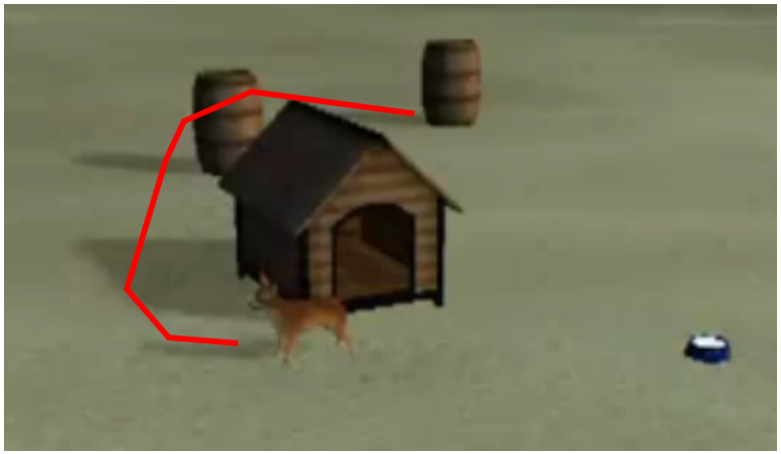
Combo interpreter actions to action plan

```
goto_obj(barrel2)
```

PAI⇓

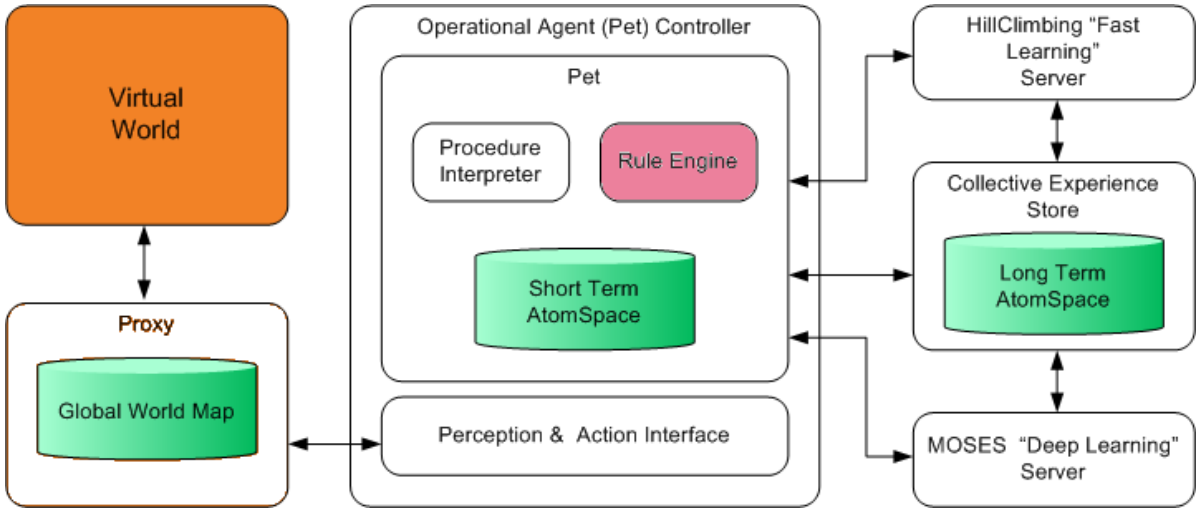
```
{walk(546, 453), walk(547, 451), ...}
```

Proxy⇓



1. Combo interpreter + PAI create action plan
2. PAI sends it to Proxy, which sends it to virtual world

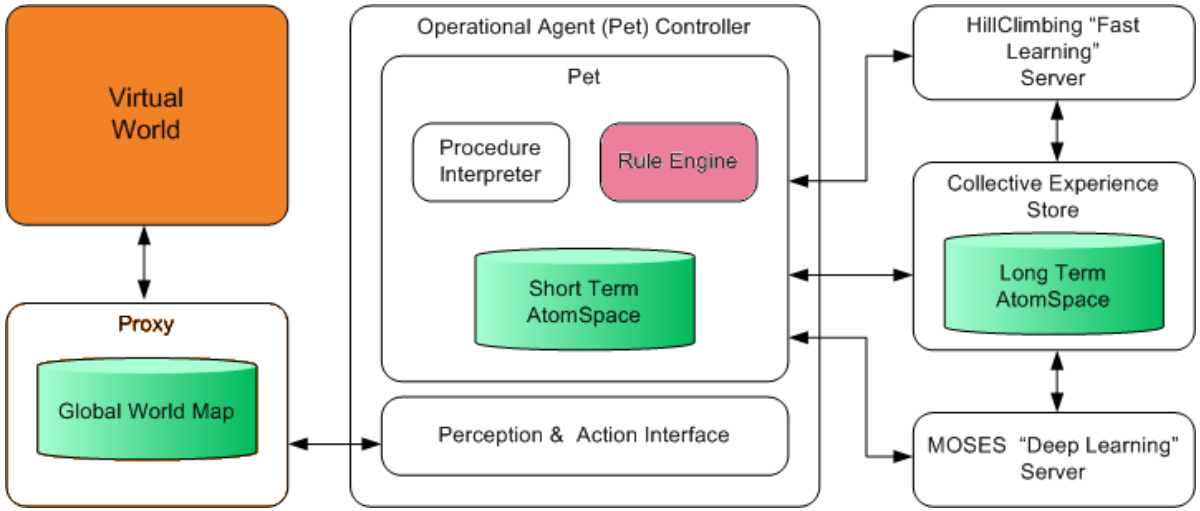
Rule Engine



Rule Engine Function

- Update agent's internal state. Example pet:
  - agent's feeling (hunger, happiness, . . .)
  - agent's relation (familiar_with, friend_of, . . .)

Rule Engine



Rule Engine Function

- Update agent's internal state. Example pet:
  - agent's feeling (hunger, happiness, . . .)
  - agent's relation (familiar_with, friend_of, . . .)
- Select next action to satisfy its goals. Example pet:
  - Fulfill needs (hunger, thirst, curiosity, etc)

Rule Engine, how it works?

For the moment it is really crude!

Rule Engine: Main loop

Evaluates a set of hard-coded rules to update state and choose next actions.

Rule Engine, how it works?

For the moment it is really crude!

Rule Engine: Main loop

Evaluates a set of hard-coded rules to update state and choose next actions.

3 types of rules:

1. Feeling rule

   `has_eaten(self)` ⇒ `happiness<0.8>`

Rule Engine, how it works?

For the moment it is really crude!

Rule Engine: Main loop

Evaluates a set of hard-coded rules to update state and choose next actions.

3 types of rules:

1. Feeling rule
   `has_eaten(self) ⇒ happiness<0.8>`
2. Relation rule
   `has_licked(self X) ⇒ familiar_with(self X)<0.9>`

Rule Engine, how it works?

For the moment it is really crude!

Rule Engine: Main loop

Evaluates a set of hard-coded rules to update state and choose next actions.

3 types of rules:

1. Feeling rule
   `has_eaten(self)` $\Rightarrow$ `happiness<0.8>`
2. Relation rule
   `has_licked(self X)` $\Rightarrow$ `familiar_with(self X)<0.9>`
3. Schema rule
   `is_hungry(self)` $\overset{0.6}{\Rightarrow}$ `goto_find_food()`

Rule Engine, how it works?

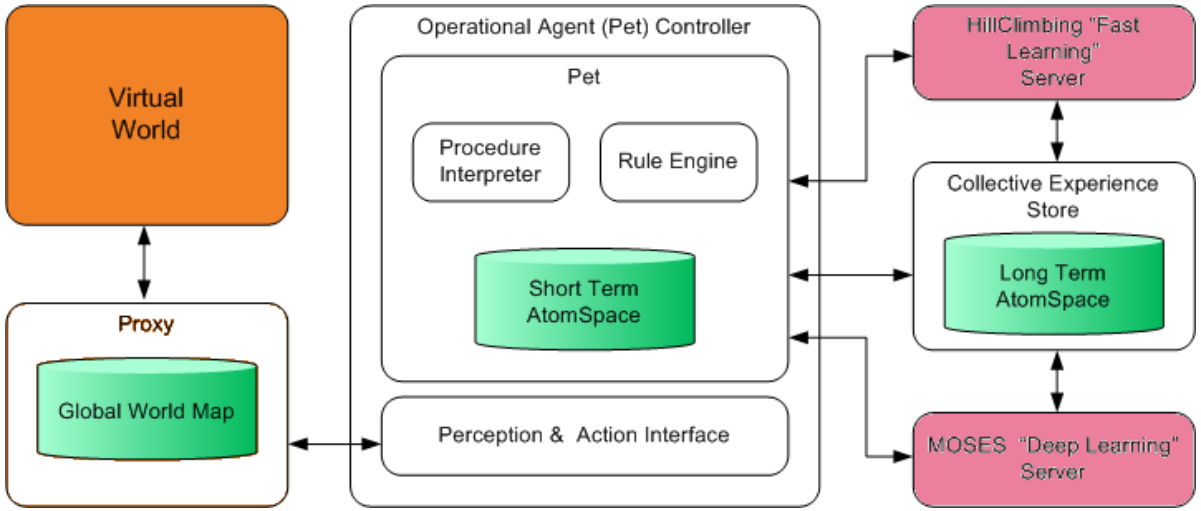
For the petBrain we have over 100 hard-coded rules!

- 21 feeling rules
- 10 relation rules
- 76 schema rules

+ rule to use tricks learned by imitation

Outline

Learning Server

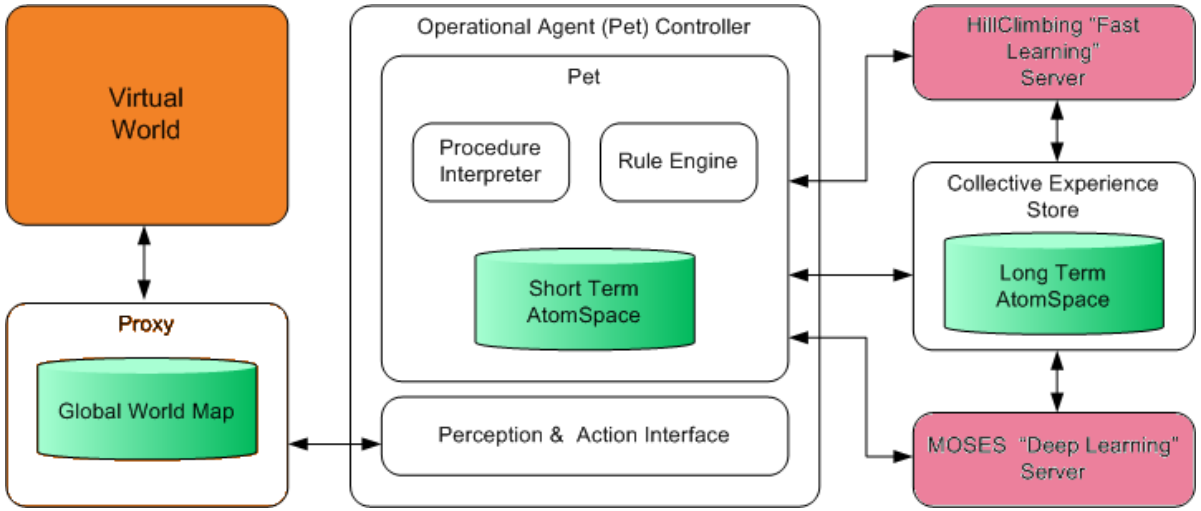


Lerning Server (LS) Function

Complex learning tasks (like imitation learning) run seperatly to not pertubate the agent's reactivity.

- Imitation Learning (Implemented)

Learning Server



Lerning Server (LS) Function

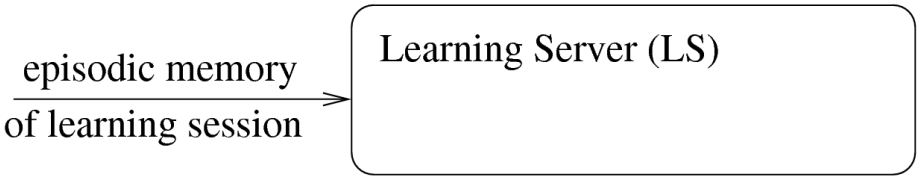
Complex learning tasks (like imitation learning) run seperatly to not pertubate the agent's reactivity.

- Imitation Learning (Implemented)
- Spontaneous Learning (not Implemented)

Imitation Learning

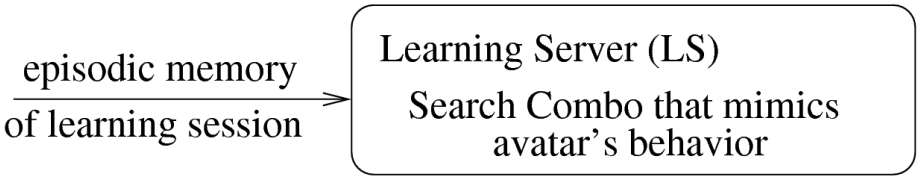
1. Pet's owner requests learning session and shows the trick

Imitation Learning



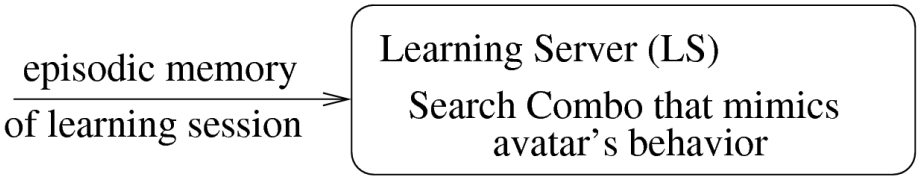
1. Pet's owner requests learning session and shows the trick
2. OPC sends portion of atomSpace, episodic memory of the learning session to LS

Imitation Learning



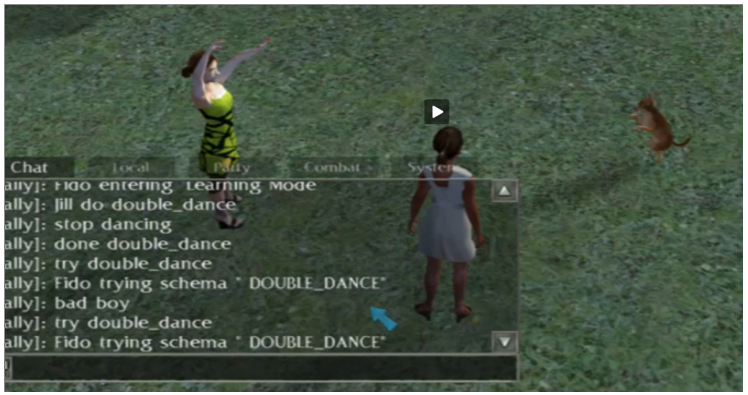
1. Pet's owner requests learning session and shows the trick
2. OPC sends portion of atomSpace, episodic memory of the learning session to LS
3. LS searches a Combo program that mimics that trick (using Hillclimbing or MOSES)

Imitation Learning



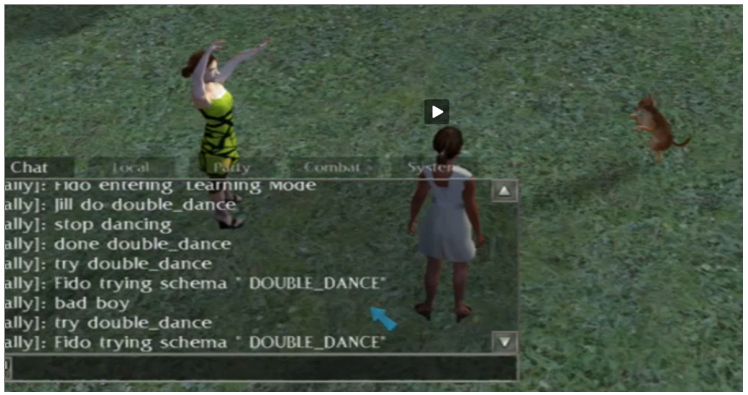
1. Pet's owner requests learning session and shows the trick
2. OPC sends portion of atomSpace, episodic memory of the learning session to LS
3. LS searches a Combo program that mimics that trick (using Hillclimbing or MOSES)
4. Each Combo candidate is run inside the imaginary world of that episodic memory, the result is compared with the avatar's behavior

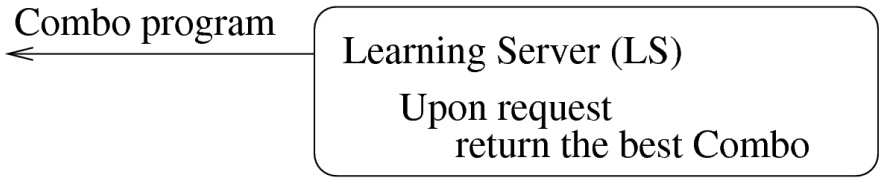
Imitation Learning



1. Owner asks the pet to try the trick

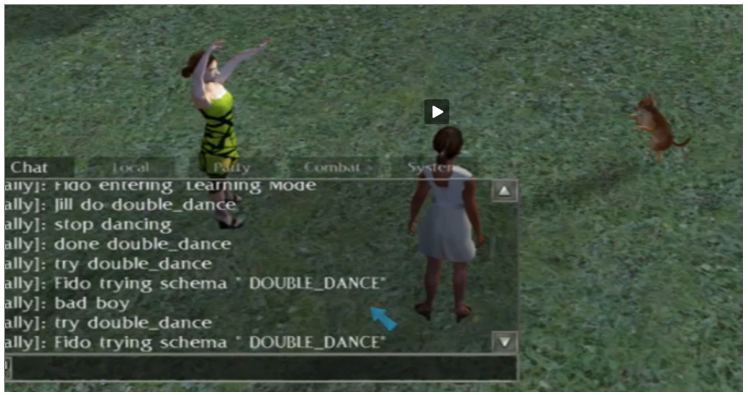
Imitation Learning

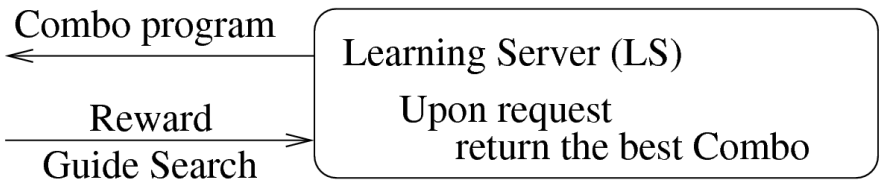




1. Owner asks the pet to try the trick
2. LS sends its best candidate so far, and pet execute the trick

Imitation Learning





1. Owner asks the pet to try the trick
2. LS sends its best candidate so far, and pet execute the trick
3. Owner sends positive or negative reward to guide the search

Example of trick learning: Double dance

Dance in loop on cue

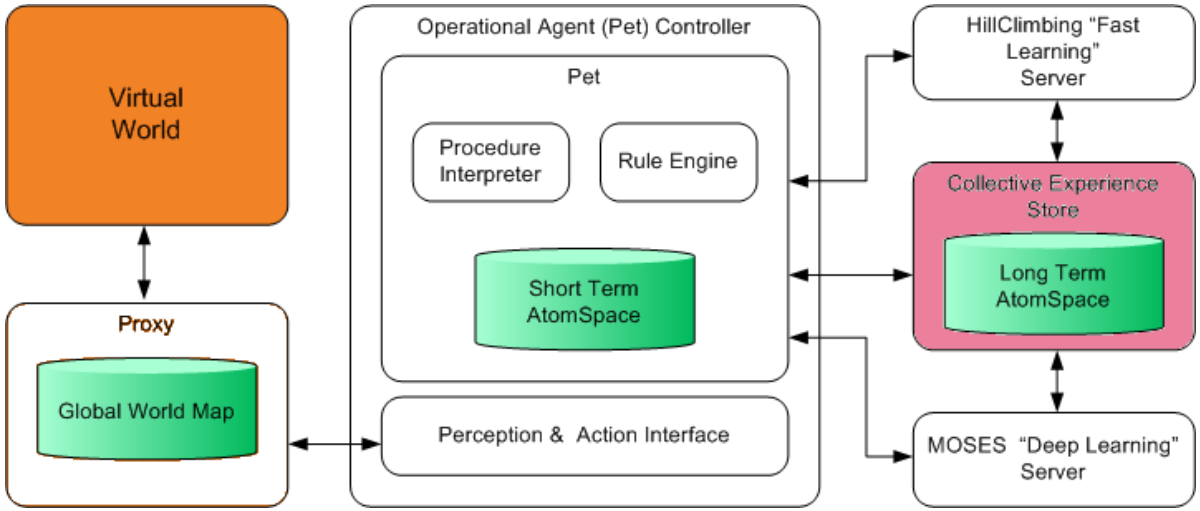
- Owner kicks left leg ⇒ tap dance
- Owner kicks right leg ⇒ lean rock dance

Combo to learn

```
boolean_while(not(says(owner "stop dancing"))
              action_if(last_action(owner "kickL")
                        tap_dance
                        lean_rock_dance))
```

Outline

Collective Experience Store (not implemented yet)



Collective Consciousness

All agents can put in common their knowledge. When one gets smarter everybody takes advantage of it.

Outline

Outline

Conclusion

What has been done:

- Hard-coded behaviors
- Imitation learning
- but no spontaneous learning
- No language understanding yet (only commands)

Conclusion

What remains to be done:

- Spontaneous learning, concept creation, integrate Attention Allocation
- Transfer learning, Collective Experience Store
- Natural language processing and generation (in development)