Integrating Reinforcement Learning into a Programming Language
Towards Usable Agent Modeling

Christopher Simpkins chris.simpkins@gatech.edu
Advisor: Charles Isbell isbell@cc.gatech.edu
AAAI-DC Mentor: Sven Koenig, USC

2010 SIGART/AAAI Doctoral Consortium
My Thesis: Integrating reinforcement learning into agent modeling can increase usability for subject matter experts (like behavioral scientists) by narrowing the semantic gap, simplifying the coding of adaptivity, and enabling modularity in agent programming.
Defining Terms

- Agent: senses its environment (state space), acts in pursuit of goals (intelligent agent). E.g., a human.
- Agent model: computer code that imitates a natural agent.
Target Users

Increase *usability* for, e.g.:
- social scientists (Gilbert & Troitsch 2005),
- game designers, and
- intelligence analysts (Sageman 2004).

\[1\] http://www.upenn.edu/pennpress/book/14036.html
Motivation

Current State of Agent Modeling Tools

NetLogo - easy but limited (Gilbert 2008).

Java (Repast\(^2\), Mason\(^3\)) - powerful but hard.

Cutting edge: ABL (Mateas & Stern 2004), A Behavior Language, created for Facade (Mateas & Stern 2003).

\(^2\)(North et al. 2006)
\(^3\)(Luke et al. 2005)
My Thesis: Integrating reinforcement learning into agent modeling can increase usability for subject matter experts (like behavioral scientists) by narrowing the semantic gap, simplifying the coding of adaptivity, and enabling modularity in agent programming.
The Semantic Gap

Information Processing | Trait-Oriented

Psychologists

Computer Scientists
Assign numeric values to each of several traits, or personality dimensions. Popular example: Five-Factor Model (McCrae & Paul T. Costa 2008).

- Openness
- Conscientiousness
- Extroversion
- Agreeableness
- Neuroticism

However, trait models very general. Key idea: multiple numeric scales.
Trait-theoretic personality models can be translated fairly directly into reinforcement learning framework.

<table>
<thead>
<tr>
<th>Psychology</th>
<th>Reinforcement Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trait</td>
<td>RL Component</td>
</tr>
<tr>
<td>Valence</td>
<td>Reward</td>
</tr>
<tr>
<td>Trait measure/score</td>
<td>Weight on RL component</td>
</tr>
</tbody>
</table>
Example: Ring Toss Experiment

- Atkinson and Litwin studied Achievement Motivation and Fear of Failure.
- 49 Students classified as high or low in both Achievement Motivation and Test Anxiety (Fear of Failure).
- Each student played a ring toss game at one of 15 distances from ring.

---

5 (Atkinson & Litwin 1960)
An Agent Model of an Atkinson Subject

object Student((Achievement, 8), (TestAnxiety, 2))

object Achievement extends AbstractRIComponent {

    world = RingTossWorld

    rewards = (1_foot_line -> 1,
               2_foot_line -> 2,
               // ...
               15_foot_line -> 15)

    actions = (play_1_foot_line,
               play_2_foot_line,
               // ...
               play_15_foot_line)
}
Agent Model Validation (Law 2007)

Lots of experimental data from psychology to validate computer models against.

- Ran 10 replications of Atkinson’s experiment (with 49 computer agents in each).
- Computer agents generated data similar to Atkinson’s human subjects.
- Conclusion: we can consider the computer agents to be good models of the human subjects (Simpkins et al. 2010).
My Thesis: Integrating reinforcement learning into agent modeling can increase usability for subject matter experts (like behavioral scientists) by narrowing the semantic gap, simplifying the coding of adaptivity, and enabling modularity in agent programming.

1. Motivation
2. Semantic Gap
3. Adaptivity
4. Modularity
Adaptivity means writing code once, adapts to new environments at run-time. Agents are adaptive by nature, but encoding Adaptivity in a language not explicitly designed for it is very hard.

- Lots of action selection code - high cognitive burden.
- *What* the agent is to do is coupled with *how* the agent should do it.
### Example: Furry Creature World. (Simpkins et al. 2008)

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>^-^-</td>
<td>oo</td>
<td>&lt;-&lt;</td>
<td></td>
</tr>
</tbody>
</table>
```

[Image of a lion and a hamburger]
behaving_entity FurryCreature {
  parallel behavior LiveLongProsper() {
    subgoal FindFood();
    subgoal AvoidPredator();
  }
  sequential behavior FindFood() {
    with (ignore_failure) subgoal MoveNorthForFood();
    with (ignore_failure) subgoal MoveSouthForFood();
    with (ignore_failure) subgoal MoveEastForFood();
    with (ignore_failure) subgoal MoveWestForFood();
  }
  sequential behavior MoveNorthForFood() {
    precondition {
      (FoodWME x::foodX y::foodY)
      (SelfWME x::myX y::myY)
      ((foodY - myY) > 0) // The food is north of me
    }
  }
}

Complex action selection logic.
A Furry Creature in $A^2BL$

behaving_entity FurryCreature {
    adaptive collection behavior LiveLongProsper() {
        subgoal FindFood();
        subgoal AvoidPredator();
    }
    Adaptive sequential behavior FindFood() {
        state {
            (FoodWME x::foodX y::foodY)
            (SelfWME x::myX y::myY)
            return (myX, myY, foodX, foodY);
        }
        reward {
            100 if { (FoodWME) }
        }
        subgoal MoveNorth();
        subgoal MoveSouth();
        subgoal MoveEast();
        subgoal MoveWest();
    }
}

What not how.
Usability Validation via User Study

Repast

```java
import repast.*;

public class MyAgentModel extends SimpleAgent {
    // Java code ...
}
```

AFABL

```java
import afabl._

object MyAgentModel extends AfablAgent {
    // RL components with states, actions, rewards ...
}
```

- “This code is a clear expression of the human being modeled.”
- “I am confident that I can learn how to write agent models in this framework.”

Perform hypothesis test on proportions of yes answers for each language. (Or maybe use Likert scale. Suggestions?)
Usability Achieved!

So integrating RL into agent modeling has *increased usability* by

- decreasing semantic gap, and
- simplifying adaptive programming.

But there’s still a problem (you may have noticed it in the example code) ...
Agents as Collections of RL Components

Recall how we defined agents at the top level:

```plaintext
// A2BL
behaving_entity FurryCreature {
    adaptive collection behavior LiveLongProsper() {
        subgoal FindFood();
        subgoal AvoidPredator();
    }
    ...

// AFABL
object Student((Achievement, 8),
               (TestAnxiety, 2))
```

An agent is a collection of RL components. Can we combine RL components? What if they’re separately authored?
My Thesis: Integrating reinforcement learning into agent modeling can increase usability for subject matter experts (like behavioral scientists) by narrowing the semantic gap, simplifying the coding of adaptivity, and enabling modularity in agent programming.
Modularity means:
- decomposition
- reuse

Existing approaches address decomposition.
- Hierarchical Reinforcement Learning (Temporal Decomposition)
  - Options (Sutton et al. 1999)
  - (Probabilistic) Hierarchical Abstract Machines (ALisp) (Parr & Russell 1998)
  - MAXQ (Dietterich 1998)
- Multiple-goal reinforcement learning (Concurrent Decomposition)
  - Q-Decomposition (Russell & Zimdars 2003)
  - GM-Sarsa (Sprague & Ballard 2003)

But reuse is a challenge ...
We’re pursuing (Bhat et al. 2006). A Learning agent, $M$, decomposed into $n$ subagents, $M = \{M_j\}_{1}^{n}$.

- Shared action set: $M_j = (S_j, A, R_j)$
- Overall state space: $S = S_1 \times S_2 \times \ldots \times S_n$.
- Overall reward: $R(s, a) = \sum_{j \in N} R_j(s, a)$ (more later ...)

Thus, the overall agent is $M = (S, A, R)$, and its joint policy $\pi(s)$ is some combination of $\{\pi_j(s)\}$.

- $Q_j(s, a)$ is Q-value of subagent $j$ for state $s$ and action $a$.
- $Q_a(s, a)$, is aggregate Q-value for the overall agent.
Existing Arbitration Approaches

Recall that $Q_a(s, a)$ is the aggregate Q-value for the overall agent. Arbitrator’s job is to find $Q_a$ and use it to select actions.

- Greatest Mass Q-Learning (GM-Q): $Q_a = \sum_j Q_j(s, a)$.
- Top Q-Learning (Top-Q): $Q_a = \max_j Q_j(s, a)$
- Negotiated W-Learning (Neg-W): subagent with the most lose selects action.

Important point: $R(s, a) = \sum_{j \in N} R_j(s, a)$ must hold, i.e., rewards must be comparable.
Our Approach: Modular Reinforcement Learning

Problem: Arbitration is social choice, therefore ideal arbitration satisfying reasonable properties is theoretically impossible.

Solution:

- Relax non-dictatorship property of ideal arbitration; other properties will still hold.
- Add a "greater good" subagent with its own reward signal.
- Learns how to optimally combine sub-agent preferences, regardless of subagent reward structures.
Our Approach to Arbitration

Arbitrator performs *command arbitration*. Formally, the arbitrator is defined by:

- Same overall state space: \( S = S_1 \times S_2 \times \ldots \times S_n \).
- An action set representing choosing a subagent: \( A = 1 \ldots n \)
- Overall reward is now “greater good”: \( R(s, a) \)

And the agent’s policy is defined indirectly by the arbitrator’s policy, \( \pi_a(s, a) \), which assigns probabilities to selection of each subagent’s preferred action for each state.

\(^6\) (Brooks 1986)
Modularity

Defining an Arbitrator

In practical terms, a furry creature would have:

- FindFood,
- AvoidPredator, and
- LiveLongProsper.

- “Greater good” - *why* avoid predator, *why* eat?
- Encodes the tradeoffs between subagents.
- Could be hand-authored, or could be another RL agent.

So, for the small cost of authoring a reward signal that represents the “greater good” you get true modularity.
Validation of Modularity

GM-Sarsa (Sprague & Ballard 2003).

Comparable rewards.

Incomparable rewards.

Our arbitrator.

Comparable rewards.

Incomparable rewards.
My Thesis: Integrating reinforcement learning into agent modeling can increase usability for subject matter experts (like behavioral scientists) by narrowing the semantic gap, simplifying the coding of adaptivity, and enabling modularity in agent programming.
Future Directions

- “Real” language
  - Current: Scala code
  - Future: Domain-Specific Language in Scala
  - Long-term: “Real” language
    - Expressiveness
    - Safety and Optimization
    - Error Messages (Usability)

- Reward authoring
  - Reward shaping
    - Heterogeneous reinforcement learning (Mataric 1994)
    - Reward signals from human feedback (Thomaz & Beazeal 2008)
Assume the current state $s$ is fixed.

At a given time step, let $U \in \mathbb{R}^{m \times n}$ be a matrix of subagent preferences. $U_{a,j}$ represents numerical preference subagent $j$ has for action $a$ and $m = |A|$ the number of actions.

$U_{a,j} = Q_j(s, a)$.

$A$ is set of all possible total orderings of elements of $A$.

Arbitration function $f$ is a function from $D \subseteq \mathbb{R}^{m \times n}$ to $A$.

In other words, $f$ takes an assignment of action preferences by each subagent and generates a social preference ordering over all the actions. In our case, “society” is the agent, and the top-ranked action is the best action for the agent as a whole to take.
Properties of Ideal Arbitration

- **Universality**: $\mathcal{D} = \mathbb{R}^{m\times n}$.

- **Unanimity**: For any pair $a, b \in A$ and for all $U \in \mathcal{D}$, if $[\forall j \in N, U_{a,j} > U_{b,j}]$, then $f(U)$ ranks $a$ as strictly preferred to $b$.

- **Independence of Irrelevant Alternatives**: For any $U, U' \in \mathcal{D}$ and subset of actions $B \subseteq A$, if $[\forall b \in B, \forall j \in N, U_{b,j} = U'_{b,j}]$, then $f(U)$ and $f(U')$ rank the elements of $B$ in the same order.

- **Scale Invariance**: For $U, U' \in \mathcal{D}$, if $[\forall j \in N$ there exists $\alpha_j, \beta_j \in \mathbb{R}$ (with $\beta_j > 0$) such that $U'_{a,j} = \alpha_j + \beta_j \cdot U_{a,j}]$, then $f(U) = f(U')$.

- **Non-Dictatorship**: Let $R_j \in A$ be the ordering over actions for subagent $j$. For all $U \in \mathcal{D}$ there does not exist $j \in N$ such that $f(U) = R_j$.

If $|A| \geq 3$, then there does not exist an arbitration function that satisfies Properties 1–5.

**Proof**: Above is a reduction of MRL to a refinement of Arrow’s Impossibility Theorem (Arrow 1963), with the addition of invariance classes, such as scale invariance, to map numeric preferences to
Universality: $\mathcal{D} = \mathbb{R}^{m \times n}$.

This property requires the domain of $f$ to be universal, i.e., $f$ must be able to handle all possible inputs. Worlds can be constructed to generate all possible settings of the $Q_j(s,a)$, thus all matrices $U \in \mathbb{R}^{m \times n}$ are attainable. We want an arbitration function that can handle all these cases.
**Unanimity**

For any pair \(a, b \in A\) and for all \(U \in D\), if \([\forall j \in N, \ U_{a,j} > U_{b,j}]\), then \(f(U)\) ranks \(a\) as strictly preferred to \(b\).

This property states that if every subagent prefers action \(a\) over action \(b\), then the arbitration function should agree with the unanimous opinion and output an ordering that ranks \(a\) over \(b\). This is also commonly known as Pareto Efficiency.
**Independence of Irrelevant Alternatives**

For any $U, U' \in D$ and subset of actions $B \subseteq A$, if $[\forall b \in B, \forall j \in N, U_{b,j} = U'_{b,j}]$, then $f(U)$ and $f(U')$ rank the elements of $B$ in the same order.

This is a requirement that the relative ordering of a subset of actions should only depend on the numerical preferences for those actions. Another interpretation is that the presence or absence of an irrelevant alternative $x \in A$ should not affect a subagent’s preference between two actions $a, b \in A \setminus \{x\}$. Arbitration functions which obey this condition are also *strategy-proof*: a subagent cannot raise the rank of an action it prefers by dishonestly reporting its numerical preferences. This effectively prevents subagents from inadvertently “gaming the system”, i.e., from learning the value of proposing an action in order to achieve better reward. This is desirable from the point of view of software reuse: a subgoal used under one parent goal can also be used under a second parent goal without fearing that the subgoal has only learned to propose actions in the presence of the first parent.
Scale Invariance: For $U, U' \in D$, if $\forall j \in N$ there exists $\alpha_j, \beta_j \in \mathbb{R}$ (with $\beta_j > 0$) such that $U'_{a,j} = \alpha_j + \beta_j \cdot U_{a,j}$, then $f(U) = f(U')$.

This property is the formal counterpart to our reward incomparability assertion. It states that a positive affine transformation of a subagent’s Q-values should not affect the ordering generated by the arbitration function. GM-Q, Top-Q and Neg-W all violate this property—under these schemes, an agent can multiply its Q-values by a large constant and completely determine the selected action. This condition essentially forces $f$ to treat the numerical preferences ordinally.

Note that requiring scale invariance allows for a more applicable theorem than requiring invariance under arbitrary monotonic transformations. Scale invariance is actually a weaker condition.
**Non-Dictatorship:** Let $R_j \in \mathcal{A}$ be the ordering over actions for subagent $j$. For all $U \in \mathcal{D}$ there does not exist $j \in \mathcal{N}$ such that $f(U) = R_j$.

This property requires that there should not exist a privileged subagent whose preferences completely determine the output of the arbitration function. On its face, this is desirable: decomposing a problem just to have a single subagent be the only one who has a say defeats the purpose of the decomposition.


**URL:** citeseer.ist.psu.edu/parr97reinforcement.html


**URL:** http://www.upenn.edu/pennpress/book/14036.html


