Designing Embedded Agents to Optimize End-User Objectives

Marcel Schoppers* and Daniel Shapiro†

* Robotics Research Harvesting
PO Box 2111, Redwood City, CA 94063
marcels@netcom.com

† Dept of Engineering Economic Systems
Stanford University, Palo Alto, CA 94304
dgs@leland.stanford.edu

Abstract. We formulate the design of discrete-state stochastic control systems as optimizing a performance objective specified in user-oriented terms, i.e. terms that need not be perceivable by the controller or agent being designed. This addresses a user acceptance issue: while agent designs (control algorithms) are limited to distinctions about state supported by artificial perception systems, end users want to evaluate performance using terms such as safety, opportunity, and throughput.[1] We elucidate a feedback from evaluation to agent design via a sensitivity analysis, obtaining the gradient of a time-averaging objective function w.r.t. state transitions influenced by the agent. This gradient leads to a methodology for iteratively improving a system's performance, as perceived by others.

1 Motivation

Although the users of artificial agents have a tremendous need to place guarantees on agent behavior, the topic of validation methodology is relatively underexplored. The empirical standard is to expose the final artifact to a large body of test cases, iteratively eliminating bugs and refining agent behavior while time and money last. The underlying performance criteria are often informal, contributing to a debatable guarantee. Verification approaches such as automated plan synthesis depend crucially on formal domain models (which are known to be inaccurate) and the availability of complete specifications (goal sets) that define desired behavior. The net result is that we are far more able to prove numerical properties of our algorithms than to validate the discrete behavior of our artifacts.

We submit that the difficulty of such validation problems stems from the need to compare reference frames: agents, designers, and users model the world in very different terms and apply qualitatively different criteria to measure success. For example, the agent (as an automaton) can only characterize the world within the limitations of its perception system, e.g. as a set of moving shapes, activated

[1] This work was supported in part by NASA contract 9-19040.

contact sensors and the like, with consequent restrictions on its decisions and behavior. The agent's engineers evaluate that behavior in terms of their own vastly superior perceptions and background knowledge, and care about such things as mechanical wear, throughput, and possible failure modes. As much as the engineers' concerns differ from anything the agent can compute, they also diverge from the interests of end users (think of bad VCR interfaces, and of pretty, but inconvenient kitchens). Users are typically motivated by questions of domain performance, convenience, and safety.

This paper introduces a design and validation methodology which explicitly addresses the need to compare reference frames. We consider a robot with a limited perceptual base that operates via a reactive loop, and an end-user whose success criteria are based on a qualitatively distinct perception of salient state. Our methodology solves the following problems:

#1 What is the mapping from robot-perceived states to human-perceived states?
#2 What is the expected utility of the robot process, expressed in end-user terms?
#3 What change to the robot's actions will best improve the end-user's received utility?

We define the mapping from robot- to human-perceived states as a probabilistic relation given a set of simultaneous observations, and ask the end-user to assign a constant utility to each user-perceived state. Next, we assume (and can compute) a predictive model of robot-perceived state, such as a Markov model. This allows us to average the end-user utility over possible futures and over time. We then derive the gradient of the expected end-user utility with respect to tunable elements of the agent's behavior. This yields a mathematical framework for evaluating and incrementally improving, the design of an artifact per human-held criteria. We submit that this expression of the validation problem is more formal than exhaustive testing and more natural than existing verification procedures.

This paper develops the necessary mathematics, and includes a walk-through of the proposed methodology in the context of a simplified NASA problem, namely the creation of a robot for space rescue. We conclude with a discussion of open issues, relevant work, and future directions.

2 Mathematics

2.1 Formalization

Our approach considers a control system, e.g. that of a robot. The robot's perceptions, including the states of its sensors and effectors and its own internal data, must be mapped into a set of mutually exclusive states S_r (mnemonic for "states perceivable by robot controller"). The number of states must be finite.

For simplicity we assume that, despite the uncertainty inherent in any real sensor suite, the agent declares itself to be in one particular state at any time. This might be the state deemed most likely in a probability distribution, or the process might indeed be fully observable. This single-state assumption is often made in discrete-state agents, which can't respond to 70% chance of rain by bringing 70% of an umbrella – the agent must act as if it's going to rain, or not. While such a single-state assumption is not necessary either for our approach or for our mathematics, it makes for a more intuitive exposition and removes an ambiguity.

Similarly for a human evaluator, observing the robot and its environment, there will be another set of states. These must be a mutually exclusive set S_e (mnemonic for "states perceivable by the evaluator"). For each state the evaluator must assess a single, constant, desirability value, and this value must depend only on the state, not on whatever comes before or after it. Clearly, the definitions of the evaluator's states will be driven by the evaluator's wish to assess different values at different times. The evaluator may however declare her belief to be any probability distribution over S_e, in which case the assessed value must be the corresponding expectation over the state's values.

S_e and S_r may be very large sets, but this does not impact the feasibility of our approach, which is intended for off-line, design-time use. (Even for $|S_r| = 10^6$ states our approach is well computable.) Further, since the evaluator need not guess at the robot's inner workings, S_e may be a much smaller set than S_r.

2.2 The Observer-Robot Relation

Toward our goal of improving robot performance as evaluated by a human observer, we need to express performance as a function of the evaluator's states, which are themselves a function of robot behavior. This means we need to derive a probability distribution over S_e for each robot state $r_j \in S_r$.

Let $p(e_i|r_j)$ denote the probability that the evaluator perceives state e_i *given* that the controller perceives state r_j. Also let $p_t(r_j)$ be the probability that the controller perceives state r_j at time t. Then by Bayes' Rule, $\sum_j p(e_i|r_j)p_t(r_j)$ is the probability that the evaluator perceives state e_i at time t.

This calculation can be written in matrix notation as follows. With $|S|$ denoting the cardinality of set S, we define a matrix $\mathbf{P}$ of size $|S_e| \times |S_r|$ whose entries are $P_{ij} = p(e_i|r_j)$. Also let $\mathbf{r}_t$ denote the vector $[p_t(r_1), p_t(r_2) \ldots p_t(r_n)]$, which is a probability distribution over the controller-perceivable states. Then we obtain $\mathbf{e}_t = \mathbf{P}\,\mathbf{r}_t$ as the probability distribution, corresponding to $\mathbf{r}_t$, over the evaluator-perceivable states. Because $\mathbf{P}$ predicts the state the evaluator will perceive as a function of the state the controller perceives, we call $\mathbf{P}$ the *evaluator prediction matrix.*

It remains only to find the actual contents of $\mathbf{P}$. From a simulation instrumented with some extra code, and at given instants of time t, we can simultaneously collect the state estimate vectors computed by the controller ($\mathbf{r}_t$) and by the evaluator ($\mathbf{e}_t$). These may identify single states, or give probability distributions over the respective state sets. Finally we compute $\mathbf{P}$ as the solution to the explicit minimization problem:

$$\min_{\mathbf{P}} \sum_t \| \mathbf{e}_t - \mathbf{P}\,\mathbf{r}_t \|^2 \quad s.t. \quad \sum_w \mathbf{P}_{wk} = 1 \tag{1}$$

where the column sum constraint guarantees $\mathbf{P}$ transforms all of the probability mass in a given robot state into *some* evaluator state. We obtain the following solution (after some algebra):

$$\mathbf{P} = \left[\sum_t \mathbf{e}_t \mathbf{r}_t^T\right] \left[\sum_t \mathbf{r}_t \mathbf{r}_t^T\right]^+ \tag{2}$$

The first bracketed term of this equation is an $e \times r$ matrix, while the second is $r \times r$, making $\mathbf{P}$ $e \times r$, as desired. (We use the pseudo-inverse (denoted by $+$) in place of the true inverse since the second term is clearly singular.) This solution guarantees that $\mathbf{P}$ is unique and that its columns have the structure of probability vectors via the following argument: since $\mathbf{e}_t\, \mathbf{r}_t^T$ and $\mathbf{r}_t\, \mathbf{r}_t^T$ are products of probability vectors, both $\sum_t \mathbf{e}_t\, \mathbf{r}_t^T$ and $\sum_t \mathbf{r}_t\, \mathbf{r}_t^T$ are non-negative matrices. $\sum_t \mathbf{r}_t\, \mathbf{r}_t^T$ is also symmetric, with non-negative eigenvalues. The inverse (or pseudo inverse) of a such a matrix has non-negative eigenvalues and thus non-negative elements as well. $\mathbf{P}$ is therefore non-negative because it is the product of two non-negative matrices. Since the pseudo-inverse operator is unique, $\mathbf{P}$ is unique, non-negative, and its columns sum to 1 by virtue of the minimization constraint. Thus, $\mathbf{P}$'s columns are (conditional) probability vectors, as desired.

Note that $\mathbf{P}$ always exists, but may be more or less informative. $\mathbf{P}$ is an identity matrix when robot and observer states are perfectly correlated, but it has constant rows when a change in belief about robot state has no impact on the estimate of evaluator state. This situation might apply if the robot and observer are on different planets.

2.3 Behavior Model Construction

To compute the expected system performance as seen by the evaluator, we use the evaluator prediction matrix $\mathbf{P}$ to map controller-perceived states into evaluator-perceived states, and calculate the system's expected behavior as perceived by the controller. For this first implementation of our methodology, we are temporarily assuming that the robot process is Markovian (possibly with hidden states). The mathematics of Markov processes is well understood, and there is a burgeoning literature on estimating the structure of Markov processes from observations, see the references in [3]. As it is trivial to make an agent's control software report the agent's perceptions of internal and external state, and thus to estimate the process's transition matrix, this paper limits discussion of the agent's behavior model to showing that the standard mathematics is indeed piecewise differentiable. See e.g. [5] for an introduction to Markov chain analysis. Here we need only the following paragraph of standard theory:

A "closed communicating class" (CCC) is a set of states that collectively function as a trap: once the process has entered a CCC it can never get out. For each CCC$_i$ the transition matrix $\mathbf{R}$ has a left eigenvector with eigenvalue 1, i.e. $\mathbf{x}_i\mathbf{R} = \mathbf{x}_i$. Rewriting the eigen-equation as $\mathbf{0} = \mathbf{x}_i(\mathbf{I} - \mathbf{R})$ we see that the eigenvectors span the null space of $(\mathbf{I} - \mathbf{R})$, and hence can all be efficiently identified with one Singular Value Decomposition. Every state having a nonzero entry in any eigenvector having $\lambda = 1$ is a member state of some (one) CCC.

Knowing which states are in CCCs, we can re-number the states so that those occurring in CCCs come first, thus yielding a transition matrix in canonical form:

$$\mathbf{R}' = \begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \Delta & \mathbf{A} \end{bmatrix}$$

where $\mathbf{C}$ covers transitions within CCCs, $\mathbf{A}$ covers transitions between transient states, and Δ covers transitions from transient states into CCCs. We can now compute the "fundamental matrix" $\mathbf{F} = \sum_{n=0}^{\infty} \mathbf{A}^n = (\mathbf{I} - \mathbf{A})^{-1}$, wherein F_{ij} is the expected number of times the Markov process will occupy transient state j when started in transient state i; and thence the matrix $\mathbf{T} = \mathbf{F}\,\Delta$ wherein T_{ij} is the probability of entering a CCC at state j when started in transient state i.

To automatically extract the transition matrix $\mathbf{R}$ from simulation traces, we assume that the controller can be instrumented with code that outputs the current controller-perceived state, as described previously. With the states identified and the possible transitions being explicit in the simulation trace, the only remaining problem is to calculate, for each state, the probabilities of transitioning to its successor states. This is a trivial calculation but for the possibility of non-stochastic behavior, e.g. a state that always transitions to itself exactly 5 times, then transitions to somewhere else. Non-stochastic behavior indicates the existence of a "hidden variable" such as a counter. An accurate Markov model must incorporate the hidden variable by modelling the repeating state as a sequence of distinct model states. We plan to use the "instance-based state identification" algorithm reported in [6], which uses state sequence observations (in addition to probability information) to automatically identify hidden variables.

2.4 The Performance Objective

We now order the human observer's subjective utilities as a vector $\mathbf{u}$, and hence predict the observer's instantaneous evaluation (at time n) as a function of the robot's initial state (distribution) $\mathbf{r}$: $\mathbf{u}^T\mathbf{P}(\mathbf{R}^T)^n\mathbf{r}$ for $n \geq 0$.

We choose an objective function that averages the (human-perceived) utility of the robot's behavior over possible futures and over time:

$$f(i) = \mathbf{u}^T\mathbf{P}\left(\frac{1}{N}\sum_{n=0}^{N}(\mathbf{R}^T)^n\right)\mathbf{r}$$

where N is a number of state transitions, possibly infinite. We choose this function for three reasons. First, we are especially interested in the design and validation of autonomous agents which act to maximize the value of an un-ending future. While goal-directed planners may tolerate arbitrarily bad outcomes that precede a goal, and ignore arbitrarily bad outcomes that follow it, an agent pursuing a time-averaged criterion behaves better in both cases. Second, time-averaged evaluation is quite general. It can be reduced to goal-directed behavior, and to shortest-path behavior, by giving all states equal utility except for one (goal) state of higher utility. Third, the above objective can be evaluated both for finite and infinite life-times, as we show next.

Rewriting the objective function $f(i)$ in terms of the canonical form $\mathbf{R}'$ of the transition matrix (see the previous subsection), and permuting the columns of $\mathbf{P}$ to match the new state numbering, we find, using $(\mathbf{R}'^T)^n = (\mathbf{R}'^n)^T$:

$$f(i') = \mathbf{u}^T\mathbf{P}' \frac{1}{N} \begin{bmatrix} \sum_{n=0}^{N} \mathbf{C}^n & \mathbf{0} \\ \sum_{m+n=0}^{N-1} (\mathbf{A}^m \Delta \mathbf{C}^n) & \sum_{n=0}^{N} \mathbf{A}^n \end{bmatrix}^T \mathbf{r}',$$

$$\to \mathbf{u}^T\mathbf{P}' \begin{bmatrix} \Lambda & \mathbf{0} \\ \mathbf{F}\Delta\Lambda & \frac{1}{N}\mathbf{F} \end{bmatrix}^T \mathbf{r}', \qquad N \to \infty. \tag{3}$$

The submatrixes have straightforward interpretations. The i^{th} row of the upper submatrix $[\Lambda\ \mathbf{0}]$ gives steady-state occupancy frequencies when the process is started in state r_i in a CCC. Similarly, since $\mathbf{F}\,\Delta = \mathbf{T}$, the i^{th} row of the lower submatrix $[\mathbf{F}\Delta\Lambda\ \frac{1}{N}\mathbf{F}]$ gives expected steady-state occupancy frequencies when the process is started at *transient* state r_i.

To evaluate a non-terminating process we focus on the steady-state behavior, namely the behavior inside CCCs as given by the left-hand submatrixes of Equation 3, since in the limit of infinite time, the time-averaged occupancy frequency of transient states (the right-hand submatrixes) is 0. To evaluate a terminating process we simply reverse the foregoing bias by focusing on the right-hand submatrixes of Equation 3, with $\Lambda = \mathbf{0}$. (Note that "goal" states must precede termination to be included in the evaluation).

If we wish $f(i')$ to give some weight to both steady-state and transient behavior, we can evaluate $f(i')$ exactly as written for some finite choice of N. With more efficiency but less confidence, we can also evaluate the steady-state and transient behaviors separately as described above, and return a weighted sum.

2.5 Gradient: ∇f

To show that our objective function is not only evaluable but also useful to guide agent design, we must provide an effective method for making agents optimize the objectives. To this end we show that it is possible to compute ∇f, the gradient vector of f with respect to the agent's internal state transition probabilities.

We obtain the elements $\frac{\partial f(i)}{\partial R_{kl}}$ by constructing a perturbation matrix $\widetilde{\mathbf{R}}$ whose entries are zero but for $\widetilde{R}_{kl} = \epsilon$. We write ${}^{\epsilon}f(i)$ for the perturbed evaluation (using $\mathbf{R} + \widetilde{\mathbf{R}}$ in place of $\mathbf{R}$). Thus we obtain:

$$\frac{\partial f(i)}{\partial R_{kl}} = (\mathbf{u}^T\mathbf{P}) \left(\frac{1}{N} \sum_{n=1}^{N} \sum_{m=0}^{n-1} (\mathbf{R}^m)_{ik} ((\mathbf{R}^{n-(m+1)})_{l*})^T \right) \tag{4}$$

$(\mathbf{R}^m)_{ik}$ is the probability that the process will be in state k, and be affected by the perturbation, at time m. $(\mathbf{R}^{n-(m+1)})_{l*}$ gives occupancy probabilities for states after l (down-stream from the perturbation) at time n. $\sum_m$ thus computes the expected effect of the perturbation on all trajectories of length n, and $\sum_n$ averages the evaluation to time N.

Eq 4, with its high powers of potentially large (but sparse) matrixes, is not recommended for computation. We isolate the gradient of the *transient* behavior by putting $\mathbf{R}$ into canonical form $\mathbf{R}'$, extracting the transient submatrix $\mathbf{A}$, computing $\mathbf{F} = \sum_{m=0}^{\infty} \mathbf{A}^m = (\mathbf{I}-\mathbf{A})^{-1}$, and thus simplifying Eq 4 to get

$$\frac{\partial f(i)}{\partial R_{kl}} = \mathbf{u}^T \mathbf{P}' (\mathbf{F}_{l*})^T \mathbf{F}_{ik} / t \tag{5}$$

where $\mathbf{P}'$ is $\mathbf{P}$ with its columns permuted to match the new state numbering. Since t scales all the gradient elements it is arbitrary. This simplification can be computed with one Singular Value Decomposition, one matrix inversion, and a few vector products. Because $\mathbf{I}-\mathbf{A}$ is very sparse, this computation can be made very fast: nearly linear in the number of states. The gradient for terminating processes of 100 states can be computed in 0.007 sec; for 10^6 states it takes 28 sec to 50 mins (on an old Sparc 2), depending on process structure.

The gradient for steady-state performance is work-in-progress.

2.6 Total Derivatives

The gradient gives crucial information for optimizing the design of the controller, specifically by focusing attention on the transition probabilities the end-user would most want changed. However, we are not free to change them arbitrarily. Our use of a Markov model imposes the constraints

$$\sum\nolimits_m R_{km} = 1, \quad \text{and} \quad \sum\nolimits_m \frac{dR_{km}}{dR_{kl}} = 0. \tag{6}$$

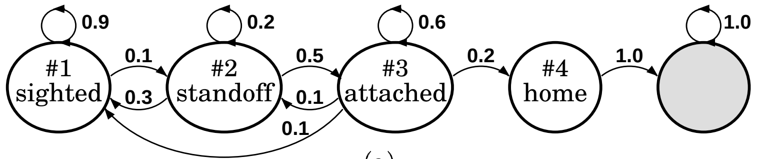
To satisfy these constraints the iterative improvement process must be concerned with the total derivatives

$$\frac{df(i)}{dR_{kl}} = \sum_m \frac{\partial f(i)}{\partial R_{km}} \frac{dR_{km}}{dR_{kl}}. \tag{7}$$

Other constraints arise from the physics of the underlying sensors and effectors, and from the economics of changing the agent design: not all conceivable improvements are feasible. Hence the revealed optimization problem is to maximize f subject to reality constraints. As usual, the optimal agent design will occur either at a boundary point of the physics constraints, or at a bliss point where the total derivatives are zero.

3 A Worked Example

The following example comes from a domain of considerable interest to NASA: a robot whose task is to rescue an astronaut adrift in space. A simplified version of the robot might be cognizant of four states: #1 astronaut has been sighted far away, #2 robot is standing-off the astronaut, #3 astronaut has been attached, and #4 astronaut has been released into the Shuttle. The actions performed in these states are: #1 navigating to an arm's length from the astronaut, #2 matching the astronaut's motion and then grappling him, #3 navigating back to the Shuttle and releasing the astronaut, and #4 deactivating the robot. Along the way, things can go wrong, like bumping the astronaut before matching his motion, or "dropping" him on the way back. A possible state diagram is shown in Fig 1a, with transition probabilities on the arcs. Since we choose to regard this "process" as terminating at a goal state, we cast it as the transient part of a Markov chain, and plan to use Eq 5. To be included in the evaluation, the goal state must be transient, so we have it transition into a new terminal state, shown shaded. The resulting transition matrix **R** appears in Fig 1b. **A** is the upper-left 4×4 part of **R**, and $\mathbf{F} = (\mathbf{I}-\mathbf{A})^{-1}$ appears in Fig 1c.



(a)

$$\begin{bmatrix} 0.9 & 0.1 & 0 & 0 & 0 \\ 0.3 & 0.2 & 0.5 & 0 & 0 \\ 0.1 & 0.1 & 0.6 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 1.0 \\ 0 & 0 & 0 & 0 & 1.0 \end{bmatrix}$$

(b) **R**

$$\begin{bmatrix} 27 & 4 & 5 & 1 \\ 17 & 4 & 5 & 1 \\ 11 & 2 & 5 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(c) **F**

$$\begin{bmatrix} 0.9 & 0.8 & 0.6 & 0 \\ 0.1 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}$$

(d) **P**

$$\begin{bmatrix} -5.4 & -4.05 & -2.7 & 2.0 \\ -0.8 & -0.60 & -0.4 & 0.3 \\ -1.0 & -0.75 & -0.5 & 0.4 \\ -0.2 & -0.15 & -0.1 & 0.1 \end{bmatrix}$$

(e) ∇f

Fig. 1. Astronaut rescue, robot & evaluation models.

The observer cares about (S) has the astronaut been returned to the Shuttle, and (H) is he harmed, yielding 4 observer states. This categorization emphasizes that the end-user is unconcerned with details of the retrieval, but does consider harm, which the robot cannot measure. The observer says that a return is worth +10, harm is worth −5, thus giving us an explicit value function, and assigning the following values:

$$\neg\text{S}\&\neg\text{H } 0,\ \ \neg\text{S}\&\text{H } -5,\ \ \text{S}\&\neg\text{H } 10,\ \ \text{S}\&\text{H } 5,$$

and these constitute the **u** (utility) vector in Eq 5.

Simultaneous sampling of robot and observer states, followed by application of Eq 2, yields the observer **P** matrix of Fig 1d, whose columns map the robot states of Fig 1a into probability distributions over the observer states just listed (in the given order). The observer appears to believe that each stage of the rescue increases the probability of harm. This might be explained by elapsed time, or by the robot itself doing harm on contact, but explanations are not essential. Multiplying out $\mathbf{u}^T\,\mathbf{P}$ yields the expected utility of *robot* states to the end-user:

$$-0.5,\ \ -1.0,\ \ -2.0,\ \ 7.5\ .$$

Finally, assuming the robot starts in state #1 and taking $t = 100$, Eq 5 yields the gradient elements. Fig 1e shows them arranged as a matrix with $\nabla f_{kl} = \frac{\partial f(1)}{\partial R_{kl}}$.

Before we can identify the most helpful changes we must recall that these are only partial derivatives where we want the total derivatives given by Eq 7. In other words, when considering which robot transition to make more likely we must also ask, at the expense of which other transition in the same row? The total derivative is the difference of those gradient elements. Thus it turns out that the largest net benefit accrues at $\nabla f_{14} - \nabla f_{11} = 7.4$ utils/step, going directly from being outbound to having returned, without any potential delays and harms. Impossible! The best realistic change is at $\nabla f_{34} - \nabla f_{31} = 1.4$, increasing the probability of the return transition by reducing the probability that the astronaut will have to be chased down again. This might be achieved by simply increasing the strength of the robot's grasp. The next best change is at $\nabla f_{12} - \nabla f_{11} = 1.35$, spending less time in the approach state. (Notice that $\nabla f_{12} < 0$, but a larger R_{12} can still yield better performance.) All other changes are much less significant.

How much can $R_{34} - R_{31}$ and $R_{12} - R_{11}$ be changed, and at what cost? Whether and how to proceed are project management decisions. Nevertheless, our approach has identified *what* is most worth improving.

4 Summary and Discussion

While most of this paper has developed mathematical foundations, its purpose has been to articulate a novel design and validation methodology for discrete-state embedded agents. We constructed an objective function over ongoing robot behavior by means of user-held criteria, and then derived the gradient ∇f with respect to perturbations of state transition probabilities deep within the agent model. Availability of ∇f led to the following design and validation methodology:

1. Derive the observer state prediction matrix $\mathbf{P}$ as described above, and obtain $\mathbf{u}$, the vector of the observer's subjective utilities for the evaluator/user-defined states.
2. Use ∇f to identify the most influential state transition probabilities in the Markov model of agent behavior, as well as whether to increase/decrease them.
3. Change the indicated state transition probabilities by redesigning the agent. ($\mathbf{P}$ need not be re-computed as long as design optimization changes only the state transition probabilities.)
4. Repeat from step 2 until...

Since the physical and financial constraints on the system's design may not be represented mathematically at design time, final validation will require an explicit decision that each of the remaining opportunities to improve total system performance is not worth the cost. The cost may be monetary, or may result from a performance tradeoff.

Reliance on the gradient of the objective function has several implications. Obviously, the numerator (the objective function) must be at least piecewise differentiable w.r.t. something the robot can influence. This requirement comes to apply to each of

(a) the expectation of robot state occurrence probabilities/counts,
(b) the mapping from robot- to user-perceived states,
(c) the mapping from user states to values, and
(d) the manner of reducing all possible futures down to a single number.

(b) and (c) will usually be linear and trivially differentiable. We adopted a Markov model of robot state for (a), and evaluated a time-averaged utility for (d), thus achieving differentiability. The important point here is that those choices are not essential to our approach: any other choices that satisfy the differentiability requirement would do.

The gradient's other implication is from its denominator: only continuous parameters of the robot's behavior can be candidates for improvement. We cannot evaluate the net benefit/loss that might accrue from choosing an entirely different reaction to a given state, or from enlarging the set of robot states, e.g. by allowing the robot to perceive a new distinction. We can however evaluate whether or not existing states should ever be entered.

Our only assumptions about the human observer are that we can sample his/her state simultaneously with that of the robot, and of course that the observer can define a set of mutually exclusive states with associated (constant) values. In short, there is no requirement on the human observer's state sequence (e.g. that it be Markovian) nor is there any restriction on the nature of the observer's utility function. Moreover, the human observer is allowed to be arbitrarily uncertain, and may declare that s/he is experiencing any probability distribution over observer-defined states (provided of course that the value assessed for such uncertainty is a probabilistic weighting over the values of the base states).

While we have calculated ∇f w.r.t. state transition probabilities inside the robot, closing the design loop requires us to know how those transition probabilities depend on concrete parameters in the robot's perception and action subsystems. Such parameters might include thresholds on subjective belief, such that certain conditions are accepted as true iff the threshold is exceeded; or motion parameters that make actions more or less likely to succeed. To compute ∇f w.r.t. such a parameter χ requires the derivatives $\frac{dR_{kl}}{d\chi}$ which are of course application dependent and hence beyond our reach. Given such derivatives, however, the corresponding augmentation of Equation 4 is trivial:

$$\frac{df(i)}{d\chi} = \frac{df(i)}{dR_{kl}} \frac{dR_{kl}}{d\chi}.$$

5 Related Work

Our work is intended as a contribution to the methodology of agent design. While methodological questions have received little formal attention as a domain of inquiry, every agent architecture makes some (possibly implicit) commitment to design strategy. Just in this volume, for example, [1] represents agent components via C++ class hierarchies and advocates an object oriented design approach; [8] highlights agent commitments and implicitly advocates a methodology for constructing agent societies based on the orderly exploration of contract nets. The work by [7] on agents with provable epistemic properties is perhaps closest to ours in feel; it suggests that end-users identify crucial domain contraints (e.g., that vases remain on tables), and derives a satisficing agent (cast as a reactive controller) via a proof process. This work shares our interest in relating agent-held distinctions to user-held observations of the world, but it lacks a representation of expected utility and therefore any explicit guidance for an iterative improvement process.

Reinforcement learning methods [3] suggest an alternate methodology for optimizing agents w.r.t user-held success metrics: reward the agent for good behavior and let a credit assignment algorithm adjust the agent's internal structure. This approach addresses the iterative refinement problem, but uses search over possible attributions (through time) in place of the tighter focus provided by our explicit gradient calculation.

Decision theoretic planning algorithms [2] [4] seek to define utility maximizing agents as an *a priori* proof problem. This work addresses the core issue of design optimality, but is currently limited to problems of restricted size. These efforts also lack attention to the issue that utility is naturally defined in user-held terms (the algorithms currently require utility to be expressed in agent-recognized states), and once again employ search (over goals and actions) in place of the more focused attention supplied by a gradient calculation. Finally, decision theoretic planning systems (as paper exercises) lack a reality check on the plans they produce. In contrast, our methodology employs end-user observations to construct a state correspondence mapping; this feedback ensures that the end-user optimizes real world performance, versus the predicted performance of an agent model.

References

1. J. Bryson. Agent architecture as object oriented design. In this volume.
2. D. Draper, S. Hanks & D. Weld. Probabilistic planning with information gathering and contingent execution. Proc Int'l Conf on AI Planning Systems 1994, 31–36.
3. L. Kaelbling, M. Littman & A. Moore. Reinforcement learning: a survey. JAIR 4, 1996, 237–285.
4. N. Kushmeric, S. Hanks & D. Weld. An algorithm for probabilistic planning. AI Journal 76:1, 1995, 239–286.
5. D. Luenberger. *Introduction to Dynamic Systems – Theory, Models and Applications*, chapter 7. Wiley & Sons, New York, 1979.

6. R.A. McCallum. Hidden state and reinforcement learning with instance-based state identification. *IEEE Trans SMC (B)* 26:3, 1996, 464–473.
7. S. Rosenschein & L. Kaelbling. The synthesis of digital machines with provable epistemic properties. Theoretical Aspects of Reasoning Conf 1986, 83–98.
8. E. Verharen, F. Dignum & S. Bos. Implementation of a cooperative agent architecture based on the language-action perspective. In this volume.