3 Background: modeling interaction between two agents with single-agent decision-making models

3.1 Markov Decision Processes

Markov Decision Processes (MDPs) are among the simplest models of sequential decision-making. They are extensively used when outcomes are stochastic (random) but influenced by a decision maker. MDPs represent the surroundings of the agent (the *world*, the *environment*) by assigning values to so-called states. The agent can take actions that change the state of the world; when doing so, it experiences costs or benefits, generically referred to as *rewards*, related to the costs of taking actions and the value of being in a particular configuration of the world.

More formally, an MDP is a collection of four objects $(\mathcal{A}, \mathcal{S}, T, R)$:

(1) The action space $\mathcal{A}$ specifies which actions an agent may perform. For example, if an agent has three possible actions: *cut*, *paste*, and *copy*, then $\mathcal{A} = \{cut, paste, copy\}$ or equivalently $\mathcal{A} = \{1, 2, 3\}$.
(2) The state space $\mathcal{S}$, specifies what the states are, and what values they could possibly achieve. For example, the state could represent the last user action and the number of keyboard shortcuts used since the start of the interaction. The state space would then be all combinations of potential user actions and the set of integers. Essentially, the action and state spaces define the inputs and outputs of the MDP, and form what could be called an "interface" or "API" of the MDP.
(3) The transition function $T(s', a, s) = \mathbb{P}(s'|s, a)$ which describes the probability of the state transitioning from state s to $s's$ when the agent selects action a. The transition function encodes the internal logic of the world, and is used to simulate the effect of actions.
(4) The reward function $R(s', a, s)$ which describes the reward the agent gets when the state transitions from s to s' when the agent selects action a. The reward function encodes the utility/value of being in a particular state *i.e.*, it evaluates outcomes.

Solving an MDP. The policy of an agent is the probability $\pi(a|s)$ that the agent selects action a when in state[1] s; essentially it is the agent's "rulebook", which describes its plays. When the agent "plays" K times according to its policy, it will gather on average a cumulative reward value known as the *return* of $\mathbb{E}_{\sim\pi}\left[\sum_{k=1}^{K} R(s'_k, a_k, s_k)\right]$[2]. We say that the MDP is solved when the agent finds the policy that maximizes the average return. Solving MDPs has received significant attention, and existing solutions can be found *e.g.*, by dynamic and linear programming [82] and (deep) reinforcement learning methods [115, 157].

3.2 Partially Observable Markov Decision Processes

A Partially Observable Markov Decision Process (POMDP) is an extension of an MDP to the case where the agent can not perfectly observe the state. Thus, instead of taking decisions based on state values, it can reason about the belief it has about the state value given its current observations. With

- Ω the space of possible observations,
- $O(o|s', a) = \mathbb{P}(o|s', a)$ an observation function, that describes the probability of producing an observation o considering the world arrived in state s' after action a,

[1] Knowing the present state is enough because of the assumed Markov property of the environment that $\mathbb{P}(s_k|s_{k-1}, s_{k-2}, \ldots, s_0, a) = \mathbb{P}(s_k|s_{k-1}a)$, hence the name Markov decision process.
[2] It is common to have an extra term called the discount factor γ in the definition of the return. This discount term is useful because when $K \to \infty$ it ensures convergence of the iterative procedures used to solve the MDP, and because it is known that people usually prefer immediate than delayed rewards. For simplicity we don't introduce the discount factor explicitly, but it can be considered to belong implicitly to R.

a POMDP is the tuple $(\mathcal{A}, \mathcal{S}, T, R, \Omega, O)$. The solution concept for a POMDP is identical to the MDP, *i.e.*, the agent has to find the policy $\pi(a|o)$ that maximizes its average return.

3.3 Using single-agent models for two-agent settings

We now consider a two-agent setting, where the agents are called U and A. As will be made clear in section 5, U stands for *user*, while A stands for *assistant*. Both agents are capable of observing the world state and producing actions accordingly. Let us imagine they are taking turns one after the other. We also assume perfect observations for simplicity. With a_U and a_A the actions of agents, the sequence of actions and states after one turn is

$$s \longrightarrow a_U \longrightarrow \tilde{s} \longrightarrow a_A \longrightarrow s'. \tag{1}$$

From the point of view of agent A, the transition function can still be expressed as $T(s', a, s)$ *i.e.*, the agent can still learn that selecting a from state s will lead to s' and is associated with reward $R(s', a, s)$. From A's point of view, U is absorbed in the world, and the problem remains a (PO)MDP.

There is however a problem: In the classical MDP setting, $T(s', a, s)$ is stationary *i.e.*, it is assumed that the world reacts the same way (barring stochasticity) every time action a is taken in state s. However, the transition $s \longrightarrow a_U \longrightarrow \tilde{s}$ in Equation 1 is in part determined by U's policy π_U. Thus, if π_U is not stationary, then neither is the transition function $T(s', a, s)$. The implication is that we lose the theoretical guarantees of algorithms that solve MDPs, which typically assume stationary transition functions, which means learning algorithms may fail to converge towards the optimal policy: in essence, the agent has to learn an ever moving target.[3]

Another difficulty of the two-agent setup is that the rewards experienced by A may not actually be due to A's choices: U may very well be doing all the "heavy lifting", and what may seem like good action choices from A could actually be poor choices. This is known as the *credit assignment* problem. So, not only is learning an optimal policy difficult due to the stationarity problem, so is evaluating that policy in the first place.

4 Modeling interaction between two agents with a multi-agent decision making model: POSG

If agent A actively accounts for the fact that there is another agent, then it can try to exploit the structure of the two-agent interaction in several ways:

(1) it can try to capture strategic interaction: the transition of the environment, instead of being driven by the agent's action a, will now be driven by both agent's actions a_U and a_A, and so it is clear that A and U's actions need to be coordinated to be efficient, instead of A treating U's actions as noise/variability. A may even learn to adjust its own actions before U's input, if it has a model of U's policy.
(2) It can try to model information and beliefs about the other. For example, it can try to model U's observation and try to deduce what U's next action could be.
(3) It can try to generalize to unseen U agents. For example, it can have a representation for policies of common subgroups of U, such as a experts and novices. Then, it can try to classify a new U as either expert or novice and then predict their action by running the appropriate policy. This technique is called determining an agent's *type* [1].
(4) It can try to perform intent recognition. For example, assuming U's intent can be parametrized by a parameter θ, and that A has access to a model of U's behavior $p(a_U|\theta)$, A can determine U's intent through bayesian

[3]The idea of a moving target may be taken figuratively, but is also very concrete in the case of TD learning, which forms the basis of *e.g.*, Q-learning, where the target is the expected return for the optimal policy.

inversion: $p(\theta|a_U) \propto p(a_U|\theta)p(\theta)$, where $p(\theta)$ is a prior on U's intentions. It can then try to act pro-actively *e.g.*, to resolve ambiguities [103].

Thus, by adopting the single-agent view, A loses the opportunity to adapt to novel behavior and to explain or predict U's actions. This implies a higher variability in the observed states, since the decisions of U add uncertainty to the already stochastic environment. Ultimately, this often leads to a loss in sample efficiency (learning takes longer) and asymptotic performance (final performance is lower). Accordingly, a multi-agent decision-making model is needed.

4.1 Partially Observable Stochastic Games

Primarily two decision-making models are used to model multi-agent interaction in sequential tasks: Extensive Form Games (EFG) and Partially Observable Stochastic Games (POSG) [126, 170]. In this work we consider POSGs because they natively support partial observations and can consider continuous action spaces [87, 90], both of which are common in HCI.[4]

A POSG (see Figure 2 for a minimal representation with two agents), is the generalization of the POMDP to multiple agents; like the POMDP, it represents the world/environment as a collection of states, and it assumes partial (incomplete, noisy) state observations and stochastic (probabilistic) state transitions. In one round of interaction, the agents *observe* the current world state, based on which they select an *action*. They do so by sampling from their own *policy*. The joint action, formed by joining all the individual agent actions, is then passed to the environment, which responds by *transitioning* to a new state, and issuing individual (agent-specific) *rewards*.

More formally, a POSG is a tuple $(\mathcal{I}, \{\mathcal{A}_i\}, \mathcal{S}, T, \{R_i\}, \{\Omega_i\}, \{O_i\})$, where $\mathcal{I}$ is the set of agents indexed by $i \in [1, \dots, n]$, and the quantities indexed by i follow from the POMDP's definition for each agent i. $\mathcal{A}$ is the joint action set $(\times\mathcal{A}_i)$ for all agents, and the joint action is $a = (a_1, \dots, a_n) \in \mathcal{A}$. A joint observation is $o = (o_1, \dots, o_n) \in (\times O_i)$. T is the state transition and observation probability: $T = \mathbb{P}(s', o|s, a)$[5], and represents the probability of the state transitioning from s to s' and the agents observing o after the joint action a when in state s. Finally, $R_i(s, a, s')$ is the reward for agent i, after the joint action a was taken in state s with the new state being s'.

A POSG (like an MDP) thus provides an unequivocal description of two agents interacting, through a mathematical interface, which specifies what the possible inputs (actions) and outputs (observations and rewards) are, and a transition and observation function, that describes the internal logic by which the system evolves to a new state from a given state after receiving the actions of both agents, and how each agent will observe this new state and experience cost/benefits via rewards.

POSG is a superset of various decision-making models, as illustrated Table. 1. MDPs are sequential single-agent decision making models where it is assumed that the agent's observation of the environment states is perfect. In the multi agent case, MDPs generalize to Markov games, or to multi-agent MDPs (MMDP) when all agents are assumed to receive equal rewards. When agent observations are imperfect, the single-agent version of POSG is the POMDP. In the multi-agent case, when all agents are assumed to receive the same reward we talk of a decentralized POMDP (Dec-POMDP).

[4]Furthermore, POSG can be viewed as the underlying model from which one derives the EFG representation in finite horizon, discrete setting [88], so in these cases there is no loss in generality.

[5]Note that it is equivalent to separate the transition function $T(s', s, a)$ from the observation function $O(o|s', a)$, as in our exposition of the POMDP.

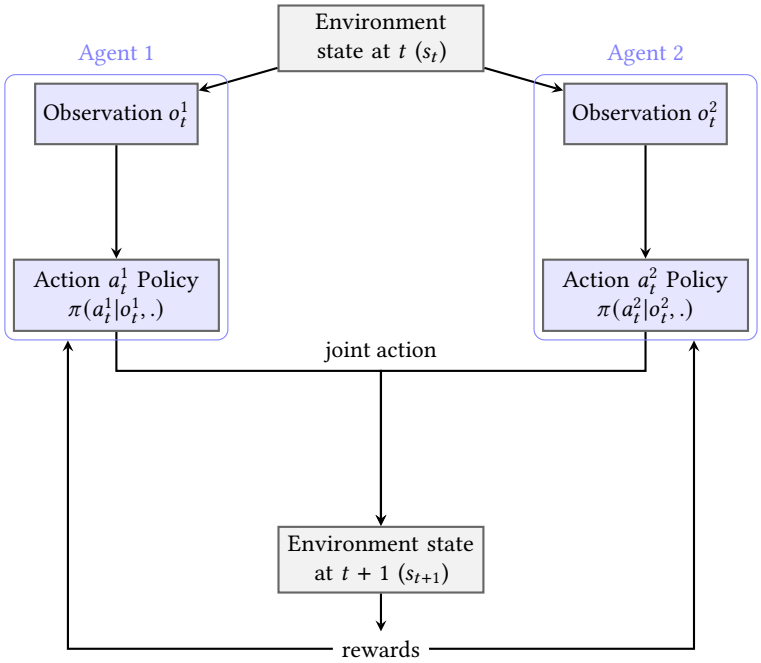

Fig. 2. Minimal representation of one cycle of a POSG with two agents. Both agents observe the present state, based on which they take an action. The two actions make the environment transition to a new state, during which it outputs rewards for both agents.

Table 1. **Some dimensions of decision-theoretic models**. The POSG, which can model partial observations in multi-agent settings where agent rewards are arbitrary, generalizes all other models showcased here *e.g.*, a POMDP is a single-agent POSG.

Observations	**Single-agent**	**Multi-agent**	
Perfect	MDP [67]	MMDP [10]	equal/shared rewards
		Markov (Stochastic) Games [154]	any reward
Imperfect	POMDP [80]	Dec-POMDP [125]	equal/shared rewards
		POSG [57]	any reward

4.2 Solving POSGs

While decision-making models can be used purely as descriptive tools, a fundamental and useful question is how to solve them. Indeed, solving decision-making models is used to model user behavior or drive the design of intelligent systems, see subsection 4.5. What it means to solve a single-agent decision-making problem has already been touched on in subsection 3.1 *i.e.*, determining the policy of the agent that maximize its expected cumulative reward during the interaction.

In the multi-agent case, it is less obvious what criterion should be achieved to declare the POSG solved: is it to maximize the cumulative rewards, averaged over all agents? Should we guarantee that the cumulative rewards gained by all agents are sufficiently close to each other *i.e.*, the game is not too unequal? One recurrent objective is to find stable (Nash) policies [119] *i.e.*, policies where no agent has any incentive to deviate; the idea being that this represents an acceptable solution for all agents, and thus towards which they may realistically converge.

Even when we specify what we mean by solving POSGs, often that solution may be computationally intractable [57]. This is due to the generality of the mathematical framework of POSGs which leads to an exponentially large solution space which can not be searched efficiently [62].[6]

4.3 Solving POSGs by reduction

This does not mean POSGs can never be used outside a descriptive purpose. Indeed, POSGs will, under some simplifying or restrictive assumptions, often be reducible to sub-classes which can become tractable [91], as shown in Tab. 1.

As an illustration, consider a two-agent POSG, with one *user* agent and one *assistant* agent (see section 5). We consider as restrictive assumption the status of the agent's policies whether known or learned, and the impact this makes on the POSG and on its potential use in HCI in Table 2:

- If both user and assistant policies are known, there is nothing to solve in the POSG; the latter is then simply a way to specify the internal logic of the interaction *i.e.*, the "rules of the game". This situation can be used for policy evaluation *i.e.*, to simulate the rewards experienced by both agents for given user and assistant policies.
- If the assistant policy is known but not the user's, the POSG reduces to one of its single-agent version, typically a (PO)MDP. Solving the POSG is then equivalent to learning the policy that maximizes the user's reward *i.e.*, producing a model of user behavior under the assumption that they are rational.
- If the user policy is known but not the assistant's, the POSG again reduces to one of its single-agent versions, but this time the objective is to find the policy that maximizes the assistant's reward *i.e.*, design an optimal assistant, where the cost function being minimized is the negative cumulative rewards of the assistant.
- The case when both policies are unknown and have to be learned is the full POSG version, and models the situation where both agents learn to behave optimally simultaneously. It can for example be used to model reciprocal adaptation [55, 95] *i.e.*, how a user learns when paired with an adaptive assistant.

Often we can't know the policy of the agent(s) exactly. Still, many other reductions remain possible. For example in Cooperative Inverse Reinforcement Learning (CIRL) [56], Hadfield-Mennel *et al.* introduce a two-agent Markov Game, but show it can be reduced to a (single-agent) POMDP. In their bounded memory model of mutual adaptation, Nikolaidis *et al.* [121] introduce a two-agent MDP (MMDP), but show it can be reduced to a MOMDP (Mixed Observation MDP, a specific POMDP with factorized state spaces, where some states are perfectly observable and others are hidden/latent) Many other sub-classes of POSG exist which make solving POSGs computationally less intensive, for example regarding observations (public observations [42, 62] *i.e.*, whether agents share some of their observations, one sided-observations [18] *i.e.*, one agent has partial and the other has perfect observations, or actions (public [42] *i.e.*, the actions of the agents are visible to each other).

4.4 Two-agent view for the modeler, single-agent view for the agent

It may seem surprising that we advocate for two-agent models by arguing what they make possible over single-agent models, but end-up suggesting single-agent reductions. The contradiction is only apparent.

From the point of view of the external modeler *i.e.*, at *design time*, explicitly modeling the human user as an agent allows naturally incorporating assumptions about the user (knowledge, goals, behavior, perception, capabilities), as argued earlier in this section. More generally, it allows describing how the environment works, what the knowledge of each agent is and how they interact together. However, from the assistant's perspective *i.e.*, at *execution/deployment*

[6] In terms of complexity, POSGs are NEXP-complete, which is worse than POMDPs which are PSPACE-complete [7].

time, the entire system can be reduced to a single-agent problem. This reduction does not invalidate the two-agent view, and in fact is necessary to make the assistance problem computationally tractable. That is, an assistant that assumes a fixed and passive environment will operate under a single-agent decision making model, but this is very different from an assistant that operates under a single-agent model that was derived from a two-agent model, because in that case the single-agent model still encodes human behavior in some shape of form (*e.g.*, latent reward parameter as in [56]). Concretely, the difference is that in the first case, the human is treated as noise/variability, whereas in the second it is treated as the source of latent variables.

Table 2. **How the POSG and some of its subclasses can be used in HCI**, depending on whether the assistant and user policies components are known (assumed prior to solving the POSG) or learned (determined by solving the POSG).

User policy	Assistant policy	
	known	learned
known	System Simulation, System Evaluation	Assistant Design (MDP, PODMP)
learned	User Modeling (MDP, PODMP)	Joint Learning (Dec-POMDP, POSG)

4.5 Suitability of POSG-type models for modeling assistance in HCI

We now discuss the suitability of using POSG-type models and their assumptions to model assistance in HCI contexts.

4.5.1 User states, and the Markovian hypothesis. Different users possess different characteristics and abilities [168]. We assume that these can be represented and quantified by a value; for example user expertise could be represented by a number between 1 and 10, or as the parameter of a parametric model. These values, when taken together, form the so-called *internal state* of the user. Often, these user states are latent *i.e.*, unobservable by the assistant and perhaps even by the user themselves. For example, most of the user's cognitive states, such as arousal, fatigue *etc.* are not observable, yet estimating these states accurately is beneficial to cooperation [66].

The decision-making models presented section 3 and section 4 make the hypothesis of Markovian state sequences[7]: that the state transitions probabilities's dependences are all encapsulated in the last visited state $\mathbb{P}(s_{k+1}|s_k, \dots, s_0, a_k) = \mathbb{P}(s_{k+1}|s_k, a_k)$. Some states of interest to us may have dependencies that last more than one state transition; for example in the teaching assistant example subsection 5.5 a more complex memory model may be used that requires information from past seen items [165]. In that case, one may simply augment the state with past states, for example if the next state depends on the two previous states, then one will consider a representation with both s_k the current state value and s_{k-1} the last state value. In many cases, the Markovian hypothesis can thus be honored simply by extending the state space, which comes at the cost of increasing the dimension of the space and makes the decision-making problem harder to solve.

[7]But not policies, otherwise learning would not be possible.

4.5.2 Observations are incomplete. User and assistant observations may be incomplete and noisy: besides latent user states, the assistant's internal state may be hidden from the user if there is no explicit desire by the designer to signal the assistant's state to the user (one example where the assistant's state is observable by the user is Octopocus [4], via feedback about the likelihood of each gesture). Users also have limited perceptual and cognitive abilities *e.g.*, decaying memory, imperfect vision, so states that are perfectly observable may actually stop to be depending on the considered time scale.

4.5.3 User behavior is reward-driven and the assumption of a rational user. The objective in the presented decision-making models is to maximize cumulative rewards; similarly in many of the solution concepts we discuss in this work when the user policy is unknown it is often assumed that it adopts a *best-response* policy. The assumption of the rational user which maximizes its utility is an old one, dating from Morgenstern and Von Neumann's work in economy and game theory, and has been challenged multiple times [81]. Still, we believe it is an appropriate framework for modeling users.

Indeed, on can always assume that users behave rationally by principle. Upon observing user behavior, the question then becomes which reward function makes the observed behavior optimal, which is essentially the inverse reinforcement learning (IRL) setup. It is known that one can construct a reward function that is optimal for any observed behavior; in fact the problem is not showing the existence of the reward function, but rather its uniqueness [76]. Thus, the rationality assumption does not necessarily reflect a normative assumption, which has been the recipient of criticism. In the IRL view, the rationality assumption is in fact infalsiable; it is not an assumption but an axiom, any behavior being possibly optimal. Thus, viewed like so, the rationality assumption is just a generic way to specify the interaction objective as the accomplishment of goals[58]. [8]

From a more practical perspective, it should also be noted that realistic agent behavior in the context of motor control, speech, perception *etc.* has been successfully modeled by considering that an agent's goal is to maximize cumulative scalar rewards [20, 68, 69, 78, 151], and that user preferences [145] and goals may also be conveyed via rewards.

4.5.4 Maximizing scalar rewards is a design strategy. Optimizing an objective cost function has shown to be a successful design strategy for interfaces [129]. The idea is simple: once the design space has been defined and constraints have been taken into account, usually many designs remain possible. It is possible to use a combination of intuition and experimentation to navigate the remaining design space and obtain good solutions, but a more systematic method is to assume the designer's objective can be cast as an objective cost function which, when optimized, returns the "best" design. This can be operationalized through assistant rewards.

While often work in HCI has focused on minimizing task completion time and error rates, in theory any designer objective can be accounted for as long as it can be encoded as rewards; for example, [19] achieve a goal of reducing the gorilla-arm syndrome, by mapping user actions to assistant rewards via a muscle fatigue model.

5 Interface Assistants Modeled as a Partially Observable Stochastic Game

In the remainder of this work, we consider as canonical situation two agents, a user and an interface assistant, that interact together, one after the other, to accomplish a task. We suggest a serial interaction model, which we show can be expressed as a POSG.

[8] To recover normative force, additional assumptions must be imposed to the reward functions. This is precisely the motivation for bounded rationality frameworks [98], which posits that human (user) behavior emerges from free choices the user makes in their best interest, within constraints imposed by the environment.

5.1 An interaction model between a user and an interface assistant

We decompose any interface into two components: a **task** — the passive part of the interface — and the actual interface **assistant**, which accounts for the decision-making part of the interface. Consider for example a typing interface with auto-correct. The task represents the part of the interface which is static, such as the (soft) keyboard and its various properties. The task's status is described by its state *i.e.*, a collection of values that specify *e.g.*, which key is active, which word is being currently suggested *etc.* The user inputs various keys when interacting with the keyboard. The assistant is responsible to determine how to interpret the keys input by the user. For example, it can choose to simply apply the keys, in which case the keyboard acts like a regular keyboard, but it can also decide to correct some of them, or suggest an entire word.

Thus, interaction with an interface assistant can be seen as the interaction between a user and an assistant who jointly perform a task, see Figure 3. In the language of POSG, the user and the assistants are the agents, while the task is the environment. For example, in the touch-typing scenario, the task is to enter a string of text by means of the keys — the state is the currently entered string. The assistant on the other hand may perform autocompletion, corrections *etc.* The assistant and the task may well be part of the same software system; what matters is what functions these entities actually perform: any decision-making is the responsibility of the assistant. It is important to note that the *user* may represent a real user *e.g.*, if evaluating a system, but it can also represent a synthetic user model *e.g.*, during *in-silico* evaluation/design.

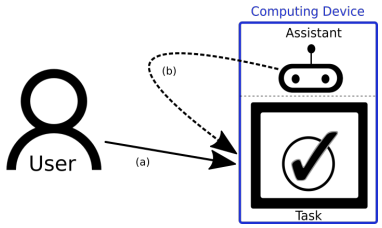

Fig. 3. A user performs a task, and is aided by an assistive interface agent.

5.2 A sequential, turn-by-turn model

The abovementioned interaction model consists of a task, a user, and an assistant. We assign each of these a state, respectively S_T, S_U and S_A. We further assume a sequential interaction model: agent updates their state and take an action one after the other, which defines 4 *turns* illustrated Figure 4:

Turn 0 The task, user, and assistant states are respectively S_T, S_U and S_A

Turn 0 → Turn 1 The user makes a partial observation, and uses that observation to update its internal state S_U to a new value S'_U

Turn 1 → Turn 2 Based on this new internal state, the user selects an action, according to its *policy*. That action makes the task transition from S_T to S'_T.

Turn 2 → Turn 3 The assistant then makes a partial observation, and uses that observation to update its internal state S_A to a new value S'_A

Turn 3 → Turn 0 Based on this new internal state, the assistant selects an action, according to its *policy*, which makes the task transition from S'_T to S''_T.

These 4 turns define a *round*; the rounds continue until the interaction finishes (usually because the task is completed).

This sequential model includes other modes that are not stricly sequential. For example, simultaneous actions may be modeled, by having the **Turn 1** → **Turn 2** transition delayed to **Turn 3** → **Turn 0**. Conditions where several user actions may be performed in between assistant actions may be implemented by having the assistant take a null or *No-Op* action, used to "skip turns".

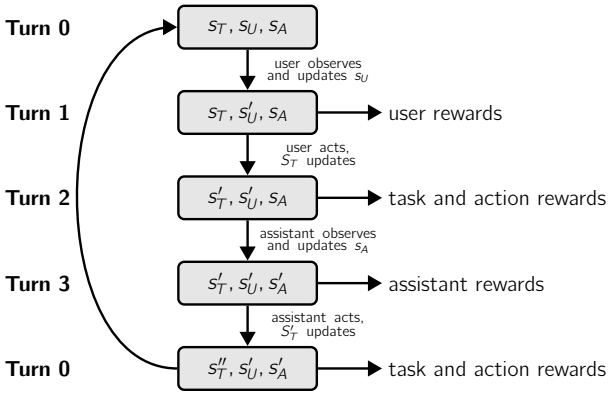

Fig. 4. Turns in a round of interaction. A round starts with the user observing the joint state, after which it updates its internal state. The user then selects an action, which makes the task transition to another state. The assistant then follows with the same turns. This repeats until the end of the interaction.

5.3 Expressing the interaction model as a POSG

The interaction model presented just above can be expressed as a POSG. The detailed explanation is delayed to Appendix B, but the crux is that the POSG state is the joint (task, user internal, assistant internal) state, and that turn-by-turn models can be made equivalent to models of simultaneous decision-making. Of direct interest here is that the transition function can be split as one external transition function that specifies task evolution and two internal transition functions that specify each agent's internal state evolution. For convenience, we also split observations from the transition function to obtain *observation functions*. In the end, the relevant functions are:

Observation functions The observation functions $p_i(o_i|S)$, which defines how agent $i \in \{U, A\}$ produces an observation o_i of the state $S = \{S_T, S_U, S_A\}$ of the entire game;

Internal transition functions The internal transition functions $p_(s'_U|o_U, s_U)$ and $p_A(s'_A|o_A, s_A)$, which define how the agent's internal states are updated based observations (respectively o_U and o_A);

Policies The policies $\pi_U = p_U(a_U|s'_U, o_U)$ and $\pi_A = p_A(a_A|s'_A, o_A)$, which define how agents select an action based on their observations and (updated) internal states.

External transition function The external transition function $p(s'_T|s_T, a_i)$, which defines how the task is updated based on agent i's action.

Each of these functions returns a reward (possibly null). The complete model is illustrated Figure 5. Note that since the agent actions are usually very informative of that agent's intent, we include agent actions as input to the observation

function. This is equivalent to incorporating them in the joint state and extending the external transition function to always update these actions.

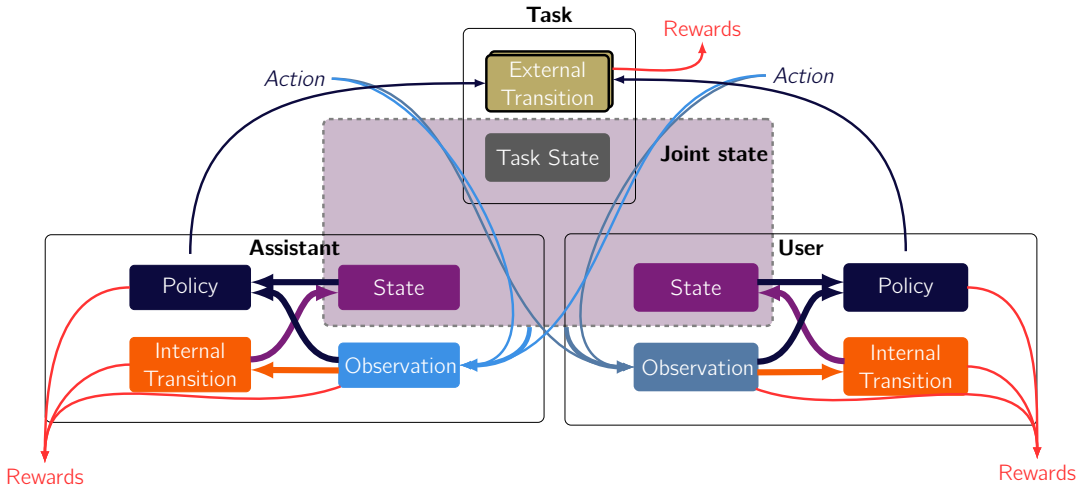

Fig. 5. The proposed model of interaction. The task and the two agents (user and assistant) are described by a state. Taken together, these form the joint state. Each agent's behavior is further described by 1) the observation function $p_i(o_i|S)$, which specifies how the joint state is perceived by the agent 2) the internal transition function $p_i(s_i'|o_i, s_i)$, which specifies how the agent's state changes after an observation 3) the policy $p_i(a_i|s_i', o_i)$, which directs how an agent selects an action, based on the agent's current state and latest observation. Finally, the external transition function $p(s_T'|s_T, a_i)$ describes how the task state evolves in response to agent actions. All functions return rewards.

We now present how a low level pointing facilitation technique and an intelligent tutoring system are expressed as multi-agent decision-making models. We specifically selected these two examples since they are classical HCI applications but not prototypical examples of agent-agent interaction, such as for example human-robot interaction, where it is obvious how the two-agent model applies. For both illustrations, we pair the assistant with a synthetic user model rather than a real user, to allow us to consensually reason about user states.

5.4 Illustration: A multi-agent decision-making model of pointing facilitation

Pointing facilitation techniques have received wide attention in HCI ([8, 9, 14, 15, 53, 97]). As a first illustration, we describe BIGPoint, a variation of BIGNav [103] without zoom. BIGPoint is a belief-based pointing facilitation technique where the assistant maintains a belief about which target is the user's intended goal.

5.4.1 *Components of the Assistance Model.*

- The *task state* represents the public information *i.e.*, the n potential targets and the cursor position $S_T = \{P, \{T_i\}_{i=1}^n\}$.
- The *transition function* details the mechanics of the interaction *i.e.*, how the cursor moves according to a CD-gain [14, 15]: $P(j+1) = P(j) + a_U \times a_A$

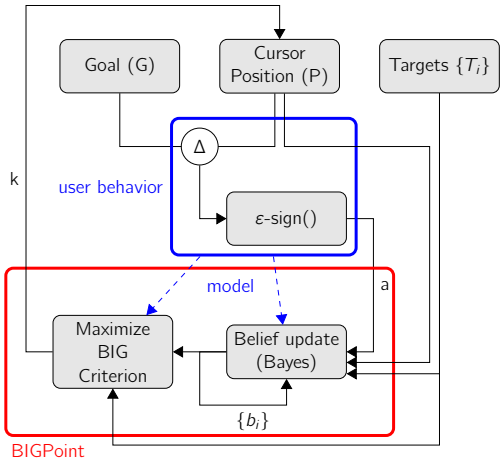

Fig. 6. **The target selection problem**. If G > P, the user selects $a = 1$, else $a = -1$ with probability ε. This action is used to update the beliefs of BIGPoint, together with a model of the user and information about the target layout. The updated beliefs are used together with the target layout to determine the next cursor position by maximizing the BIG criterion.

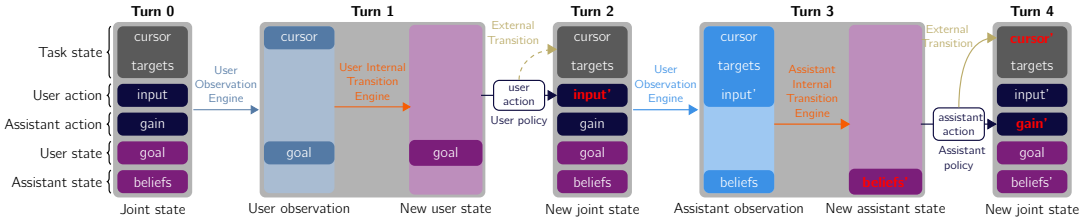

Fig. 7. **A round of interaction in the pointing facilitation example**. The game state for any given round (turn 0) is displayed left. During the first turn, the user produces a partial observation of that state; its internal state remains unchanged, since the user's goal does not change during the task. The user then selects an action, based on its policy and the game state is updated (turn 2). In the third turn, the assistant produces a partial observation of the game state, the user goal being latent. It uses that observation to update its belief distribution, which is used by its BIG policy. Bold state values indicate that their values has changed, and the green arrows indicate inputs and outputs of the key functions of the user assistance model. [CJ: This figure is quite hard to read, mostly due to the color choices]

- *User actions* are {left, right} which we represent as $a_U \in \mathcal{A} = \{-1, 1\}$. Thus $a_U = \text{sign}(\text{Goal position} - \text{Cursor position})$. We assume as user model one which makes mistakes with probability ε mistake probability, thus $\pi_U(A = a) = \text{sign}(\text{Goal position} - \text{Cursor position})$ with probability ε, else $\pi_U(A = a) = -\text{sign}(\text{Goal position} - \text{Cursor position})$.
- The assistant selects the CD-ratio a_A, to maximize a criterion proper to BIG [103]
- With the goal of minimizing the number of user actions, the *reward* is simply -1 on each round for both the assistant and user.

- The *user's state* contains the goal target $S_U = \{G\}$.
- The assistant maintains a *belief state* $\{\{p(T_i)\}_{i=1}^n\}$ over all targets, obtained by Bayesian updating
- each agent is able to perfectly *observe* the task state and their own, but not the other
- The assistant has an internal state transition to update its belief state

A round of interaction using the POSG model is represented Fig. 7.

5.4.2 Classic pointing facilitation techniques as different policies. The above example described BIGPoint, a specific pointing facilitation technique; but the description easily generalizes to other techniques: By assuming different forms of assistant policies, we retrieve many existing pointing facilitation techniques from the HCI literature.

Constant CD Gain is recovered with the assistant policy $\pi_A = K_0$, which always applies the same CD-ratio K_0. This strategy can be selected quite easily on some OS's by end users, but is usually not the default setting [14, 15].

Non-linear CD Gain is recovered with the assistant policy $\pi_A = f(a_U)$, where f is an arbitrary deterministic mapping of the user action a_U. This is by far the most common pointing facilitation technique, used on most mainstream OS's [14, 15]. The benefit of having such a policy is twofold: it is extremely simple to implement, and the deterministic nature of the mapping implies users quickly grow accustomed to it. The mapping may be adaptive, such as in Autogain [97].

Target Aware techniques Many researchers have tried to take advantage of the extra information that is contained in the target location to enhance pointing, by having assistant policies of the type $\pi_A = g(\{T_i\}_{i=1}^n, a_U)$, where the gain is picked based on the potential targets and user actions. Object pointing [53] and semantic pointing [8] are two well known examples (also see [74, Table 1] for more examples).

Goal Aware techniques BIGPoint is an example of goal aware techniques, where the assistant policy is of the type $\pi_A = h(\{T_i\}_{i=1}^n, \{p(T_i)\}_{i=1}^n, a_U)$ which further reasons on which target the actual goal might.

In the more general case, the assistant policy can make use of all the information present in the joint state, which could also further be extended, for example by using extra sensing techniques such as eye-tracking or providing more target-relevant information, such as recency and frequency data. The assistance model just presented will account for all these potential future cases, simply by appending the corresponding states with the new information to the description.

5.5 Illustration: A multi-agent decision-making model of artificial teaching

In this example, we cast the model by Nioche *et al.* [122] on artificial teachers for learning foreign languages as a multi-agent decision-making model. In their work, an assistant proposes an item among N possible (a word in the foreign language) and the user has to respond by the equivalent word in its native language. If the response is false the teacher provides the correct answer.

The objective of the teacher is to create a sequence of items that maximizes the number of learnt items at some planned date. The imperfect memory of the user makes planning the sequence tricky: suggest too many new items and overall retention will be poor, but suggest too few and the number of items learnt will be lower than what is achievable. To solve this problem, Nioche *et al.* introduced two modules: a *psychologist* module, which estimates the user's probability of recall for each of the N items, and a *planner* module, which plans the optimal sequence of items to maximize recall for a future exam based on these estimated recall probabilities, see Fig. 8.

5.5.1 Components of the Assistance Model.

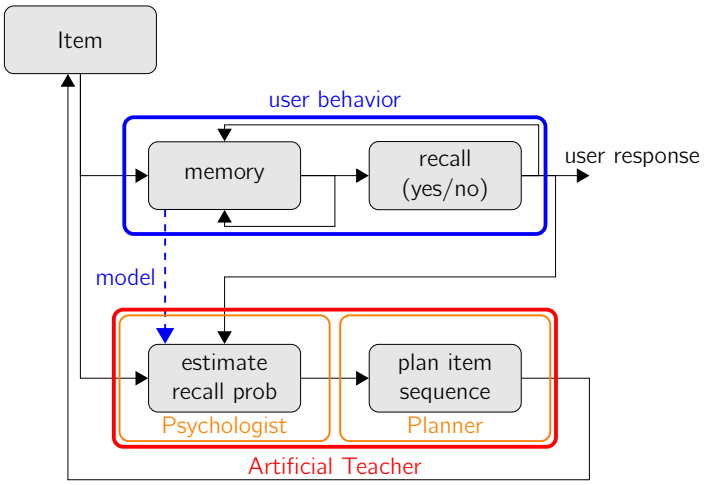

Fig. 8. **The artificial teaching problem, and Nioche *et al.*'s solution**. When an item is presented, the memory of the user is solicited and the user recalls the item or not. The artificial teacher has a psychologist module, that estimates the user's recall probabilities at each time using a memory model. It also has a planner module, which constructs the sequence of items to be presented based on the psychologists' output.

- The *task state* holds the current presented item and some bookkeeping information (bkp) as required for memory models (a time vector that tracks when an item was last presented, the current time *etc.*
- The *transition function* simply updates the newest presented item and the bookkeeping information
- *User actions* are binary (recall the presented item or not). The recall probability associated with each item is determined by a memory model (in [122] an exponential forgetting model is used).
- The *assistant action* consists of the next item to be presented to the user.
- *Rewards* are issued only once, at the end of the sequence of items.
- The *user state* contains the vector of recall probabilities p associated with each item *i.e.*, it is a representation of the memory.
- The *assistant state* contains the estimated vector of recall probabilities *i.e.*, it is a model of the user's memory.
- The user is capable of introspection (*i.e.*, it can *observe* its own p)
- The assistant *observes* everything except p

A round of the assistance model is displayed Fig. 9.

5.5.2 Different teaching strategies as different policies. The artificial teaching example generalizes several artificial teaching techniques:

Random teaching If the assistant's policy $\pi_A(\text{item}) = 1/N$, then the teacher simply randomly selects an item. This strategy does not account for a user's memory, yet can have reasonable performance if the recall of the user is good since this policy ends up presenting many items.

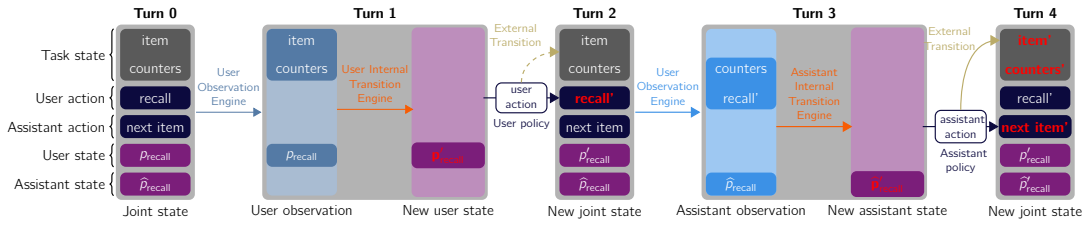

Fig. 9. **A round of interaction in the artificial teaching example**. The game state for any given round (turn 0) is displayed left. During the first turn, the user produces a partial observation of that state, and the recall probabilities are updated (memorization). The user recalls the item or not, and the state is updated (turn 2). In the third turn, the assistant produces a partial observation of the game state; it tries to reconstruct the latent user recall probabilities $\hat{p}$. These estimates of recall probabilities are used by a planner (conservative sampling policy) to determine the next item to be presented, after which the game state is updated again. [CJ: Same as above: This is really hard to read, mostly due to the color choice. The actual content is fairly clear.]

Spaced repetition With spaced repetition systems, the assistant policy is a rule-based statemachine system that takes as input the last user response *e.g.*, the Leitner system [122, 139].

Modeling user recall This class refers to assistants whose policies build on estimating the recall probabilities of the user $\hat{p}$ which are then used for planning as in [122].

These two examples highlight several benefits of the proposed description: While the pointing facilitation and artificial teaching examples are very different problems, we represent them without ambiguity with the same elementary mathematical objects of our assistance model, rather than using constructs that make sense only in a particular context *e.g.*, the *psychologist* and the *planner* in the artificial teaching scenario. This decomposition also implies modularity, as the same description generalizes to a large set of different interactions simply by changing assistant policy.

5.6 other examples

Table 3.

6 User Modeling in POSGs: A Central Challenge

In many successful interactive systems, such as search engines or recommender systems, there is no explicit model of the user needed (although, see our taxonomy next). These systems succeed by exploiting large-scale statistical patterns and optimizing for generic objectives (*e.g.*, relevance or similarity); this works because there is mostly no need to infer hidden internal states such as goals, preferences, or cognitive state of the user to determine that one recommendation is similar to another.

However, in POSGs, agents must make decisions under uncertainty and partial observability. As reminded subsection 3.2, agents with partial observability cannot reason about the environment's state; instead they must maintain a *belief* state. This implies that agents can only try to act optimally based on what they believe to be the true state, given their imperfect observation. Thus, in POSG in general, user modeling becomes essential for the assistant, so that it can coordinate, assist, or share control with a human. Some examples include:

- The assistant is usually without knowledge of the user's internal states, yet human decision-making is dependent on the latter *e.g.*, goals, knowledge, emotional states.

Table 3. Example of interfaces that are covered by our model. For each application, we give examples of one or several assistants and their actions, as well as the general goal of the users and the actions they have at their hand in typical tasks.

Application	Assistant Actions	User goal state, rewards	User actions	Examples
Target Selection Facilitation	Tune transfer function [97], alter selection mechanism [47, 52], move cursor or view [103]	Cursor on target	Mouse/keyboard input	
Intelligent text entry	Auto-completion, Auto-correction [43]	string of text correctly input	Keyboard input, accept/decline recommendations	
Adaptive keyboard	Change keyboard layout [143] or triggers [112]	string of text input faster/with less effort	Keyboard/gaze input	
Artificial teaching, intelligent tutoring	select teaching sequence [122]	maximize recalled items	recall item	
Alerting/notifying	Alert user of incoming information [70, 84]	user informed with as few false alerts as possible		
Raising user awareness	highlight, change contrast [26]	important information accessed quickly	gaze, move cursor	
Computer-aided decision making	Provide, curate, synthesize information [156]	maximize accuracy	interpret assistance to make a decision	
Artificial coaching	nudging/motivating user [79]	start the task		
Haptic guidance	add physical interaction	accomplish task		[117]
Interface design	determining layout of commands			[27, 161]
Visual design	select layout, style components			
Creativity support				[77, 86]
Desktop assistant				

- Some humans signals are ambiguous, in which case a model of the user's goals or policies may help disambiguate. For example, in pointing, it is possible to determine almost from the onset of movement which zone of the screen will contain the endpoint of the movement, yet it is very hard to determine which target in that zone the user aims for [171]. Disambiguation is possible by comparing possible user goals with target semantics.
- Personalization and adaptation is facilated by modeling user preferences, habits *etc.*

Intuitively, the less the humans intentions capabilities and constraints are observable yet significantly impact outcomes, the more user modeling becomes important. While not explicitly represented in POSG, agent modeling has also received much attention in the decision-making community [1].

Within our two-agent view, both agents are capable of forming internal models:

- Humans are well known to form internal models for many low-level tasks, such as perception and motor control [37]. For example, in visuo-motor control, human subjects maintain a forward model of the limb that fills in missing sensory information [113], a mechanism that is thought to be at play in the detection of hand-redirected movements in VR [48]. More generally, users are known to spontaneously form mental models of the software/systems they interact with [64, 123].
- Internal models of the user are an integral part of many multi-agent system design [1] and are argued by some to be necessary [16] to successful interaction with users; they namely allow the assistant to form predictions about future user actions, which facilitates coordination and alignment with the user. Internal models are also used for inference *e.g.*, as a normative assumption [3] to interpret observed behavior.

6.1 Taxonomy of user modeling

We suggest a simple taxonomy of user modeling, which considers which of decision-making model attributes are estimated by the assistant. Figure 10 describes the full POSG model, as well as the various user models that the assistant can handle.

- in *user state modeling*, an estimate $\hat{S}_U$ of the user's internal state is produced. Usually, these will be latent cognitive states *e.g.*, memory [122] or goal states [103]. The latter is common in HCI, and may be treated distinctly, because "people interact with interfaces to accomplish goals" [153]
- in *observation modeling*, an estimate $\hat{o}_u$ of user's observations is produced; it characterizes how the user perceives its environment. Such models are useful to describe the imperfect perceptual capabilities of users. For example, an asymetric noise model may be used to account for the anisotropy of foveal vision [114].
- *policy modeling* where $\hat{\pi}_u = p(\hat{a}_U | \cdot)$ is estimated. The policy can then be exploited to predict user actions. For example, a popular model in multi-agent systems is fictitious play, where the frequency of past actions is counted [116], and used to predict future actions. Sometimes, it may be more convenient to directly predict user actions $\hat{a}_U$.
- *reward modeling* where the user rewards $\hat{r}_u$ are estimated.

Before moving on, we make several remarks:

- Different fields place emphasis on different types of user models. A popular method in HCI is reward modeling, where researchers build on cognitive and perceptual models to form estimates of task completion time, which are then treated as costs (negative rewards). For example, [161] discuss an adaptive system which forms an estimate of average cumulative selection time in a menu, which the assistant then seeks to minimize.
- There is no single way to model some phenomena. For instance, since the user observation is the result of applying its observation function the state $o_U = o_U(S)$, some lossy observations can be modeled either via $\hat{S_U}$ or $\hat{o}_U$. For example, in the context of spaced repetition [122], the task state may hold all information related to past presented items. Imperfect user memory can then be modeled as the internal update of a state which loses some information, but could also be modeled as an imperfect observation of the task state.

- user models are often used in conjunction. For example, a reward model used with a matching rule [107], such as softmax [141], is a popular choice to predict stochastic user actions. A goal-conditioned policy model can be inverted with Bayes law to get a goal model, as in [103]. In inverse reinforcement learning, a reward model of the user is learned based on observed user behavior, which is then used to predict behavior through (forward) reinforcement learning [172].
- models can be learned on or off-line. In HCI, there is a longstanding tradition of working with models from the psychological sciences that are established off-line through controled experiments, but whose parameters may be continuously estimated on-line. HCI has also turned to machine learning methods, where user models are typically learned from data collected more naturalistically, either off-line or on-line.

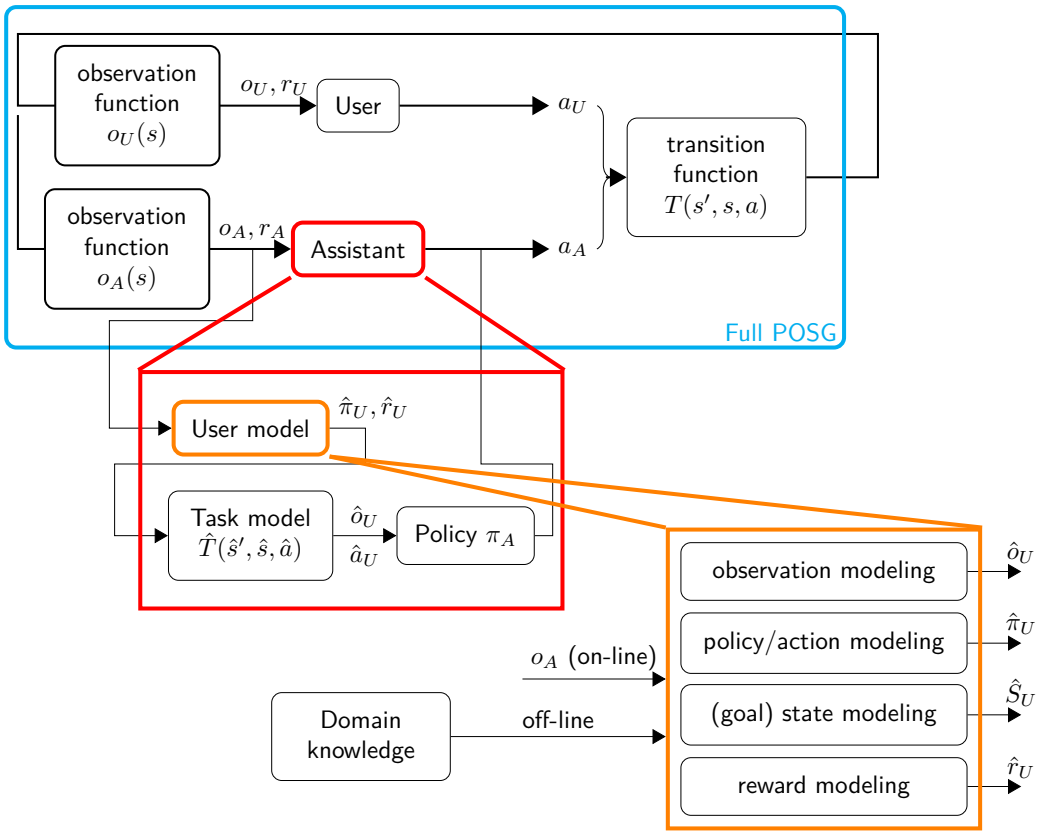

Fig. 10. Taxonomy of user modeling based on the POSG properties.

7 Different assistance algorithms

[In this section, there is a progression, from non-agentive modeling to two agent modeling methods.]