A decision-theoretic representation of assistive interfaces

Julien Gori
Sorbonne Université, CNRS, Inserm, Institut des Systèmes Intelligents et de Robotique, ISIR
Paris, France
julien.gori@cnrs.fr

Aurelien Nioche
University of Glasgow
Glasgow, United Kingdom
Aalto University
Espoo, Finland
nioche.aurelien@gmail.com

Christoph A. Johns
Applied Artificial Intelligence, University of Oldenburg
Oldenburg, Germany
Aarhus University
Aarhus, Denmark
christoph.johns@uni-oldenburg.de

Antti Oulasvirta
Department of Information and Communications Engineering, School of Electrical Engineering, Aalto University
Espoo, Finland
ELLIS Institute Finland
Espoo, Finland
antti.oulasvirta@aalto.fi

Abstract

Assistive interfaces, such as recommendation engines, adaptive systems, and intelligent assistants, span diverse methods and disciplines but lack a shared conceptual foundation. This paper models assistance as sequential decision-making under uncertainty between two agents: the user and the assistant. The formalism allows casting assistance as an optimization problem and offers a rich but principled vocabulary to understand the dynamics of assistance. Drawing on Partially Observable Stochastic Games (POSGs) and related models, we: (1) motivate multi-agent over single-agent formulations; (2) adapt POSGs to HCI and clarify their tractability through reductions; (3) propose a two-agent sequential model that unambiguously defines concepts such as adaptation, augmentation, and delegation; (4) illustrate applicability through domain problems and examples; and (5) offer a supporting implementation via a library. These results warrant more attention on decision-theory as a principled yet actionable approach to assistive interfaces.

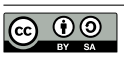
CCS Concepts

• **Human-centered computing → HCI theory, concepts and models**.

Keywords

POSG, cooperation, theory, decision-making model, library

ACM Reference Format:
Julien Gori, Aurelien Nioche, Christoph A. Johns, and Antti Oulasvirta. 2026. A decision-theoretic representation of assistive interfaces. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26), April 13–17, 2026, Barcelona, Spain.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3772318.3791819



CHI '26, Barcelona, Spain

ACM ISBN 979-8-4007-2278-3/2026/04
https://doi.org/10.1145/3772318.3791819

1 Introduction

Assistive interfaces, also referred to as *interface agents* or *intelligent assistants*, aim to improve user experience by adapting autonomously to user needs [77, 103]. These systems often observe user behavior and take action on behalf of users, creating new forms of interaction in which control is shared between the user and a computational agent. Examples include recommendation engines that suggest content based on interaction history; adaptive interfaces that adjust layout or behavior to suit user abilities; and intelligent assistants that automate or suggest task-relevant actions. Other systems, such as context-aware applications or intelligent interaction techniques, aim to support user goals by adapting in real time to perceived context and intent.

Presently, there is no unified conception of assistance in interfaces [115]. Despite the shared objective, research is fragmented across systems and studies, even fields (AI, HCI, cognitive science, design). Existing technical approaches are diverse, in terms of terminology, but also goals and methods, ranging from hand-crafted rules to model and data-based approaches, from supervised to reinforcement learning, and now to foundation models like LLMs. For instance, a recommendation systems researcher might optimize for click-through rates using bandit models [106], while an adaptive interface designer may focus on minimizing task completion time by combining behavioral models of human performance and reinforcement learning [114], making it difficult to compare which approach better serves user goals or to transfer insights between domains. This diversity reflects innovation, but also a lack of shared structure, which makes it hard to compare existing systems, understand their behavior systematically, re-use existing systems as baselines, re-use successful solutions from other interaction domains, reflect on the shared difficulty of certain interaction scenarios *etc.* We argue that a common language of the analysis of assistance — or, as Rogers [100] puts it, a *lingua franca* — is needed. It would help understand the assistance problem beyond the highly contextual characteristics of individual applications [75], emphasizing the complex dynamics of interaction and assistance, and as a result, improving understanding and collaboration between researchers from different disciplines.

At the same time, to be genuinely useful, the analysis should lead to models that can actually be implemented in software.

In this paper, we propose a decision-theoretic model of assistive interfaces and illustrate its expressiveness and utility. The core insight is to split assistive interfaces in two parts: a passive, *user*-facing component called the *task*, that predictably changes according to user actions, and an active component, called the *assistant*, that must repeatedly select actions that reconfigure the passive component with the aim to improve the user's experience. The user and the assistant, called agents, evolve in the shared environment of the task, which they can perceive, act upon, and within which they experience costs and benefits. To make decisions, the assistant must rely on their observations of the state of the task and the user, and the beliefs they maintain about the user (goals, states, capabilities). They may ground these decisions in models of the interaction task, or of the user behavior, perceptions, and intentions. At the same time, the user perceives the actions of the assistant and can, if necessary, change their behavior to work with and around their decisions, *e.g.* encouraging a correct action or correcting an overeager suggestion.

Our model explicitly represents what the system can do, what the user can do, what each one wants, and what counts as a success. It further makes explicit the user and assistant's uncertainties about the task, the environment, and the actions and goals of the other. This allows to compare and contrast different assistance approaches, and to simulate "what if" scenarios about the outcomes of different user behavior patterns, assistant strategies, or dynamics of the interaction task. It further provides a common, computationally implementable language to talk about user goals, system actions, and interaction outcomes. It also permits formally defining assistance concepts like adaptation, augmentation, or delegation, that cuts across specific applications and assistance domains.

Technically, we draw on established concepts from sequential decision-making, such as Markov Decision Processes (MDPs) [98], Partially Observable Markov Decision Processes (POMDPs) [61], and Partially Observable Stochastic Games (POSGs), to create a new turn-by-turn interaction model. This model is formally equivalent to the definition of a POSG but explicitly accounts for user and interaction task models, improving the fit with HCI conceptions of assisted interactions. This abstraction supports formal reasoning about goals, feedback, hidden context, and long-term effects of assistive behavior, regardless of the specific application. By explicitly representing key elements such as user actions, system policies, observations, rewards, and user models, our analysis enables HCI researchers to articulate and analyze the space of possible interactions and assistive strategies. The model also affords implementation; thus mapping from this model of assistance, we developed CoopIHC, a software library that illustrates the model's generalization and which we use to implement worked-out examples.

For example, consider designing an intelligent email client that assists users in managing their inbox. Traditional design methods might observe user frustrations, identify pain points, and suggest interface tweaks or automation features. Alternatively, we suggest modeling the interaction with a decision-making model: the system does not fully observe the user's true goals (*e.g.* prioritizing urgent work emails versus social messages), but must infer them from observed actions. It should choose assistive actions accordingly (*e.g.* suggesting replies, sorting messages) to maximize the user's supposed reward function, aligned with those inferred goals. We believe that this type of formalization can help researchers explore alternative assistive policies or designs, evaluate trade-offs under uncertainty, and simulate long-term behavior, all before building specific solutions and having a fully implemented system.

Our contributions are as follows:

- We explain that, while a single-agent model can be used to model assistive interfaces, this solution loses the opportunity for the system to adapt to strategic behavior, and to explain or predict user actions. This motivates the use of multi-agent decision-making models instead.
- We introduce a multi-agent decision-making model known as the Partially Observable Stochastic Game (POSG). We clarify what it means to solve such models, highlight their computational intractability, and show how reductions map POSGs to tractable subclasses. We also show that POSGs and their underlying assumptions are compatible with the characteristics of HCI. We further explain how assistance strategies obtained by POSG reductions differ from approaches that are initiated from single-agent modeling.
- We suggest a new sequential two-agent model, where a human user and an assistant agent interact together, one after the other, to jointly accomplish a task. We show that this model is equivalent to a POSG and derive four core components: observation functions, internal transition functions, policies, and external transition functions. We also show how this model accounts for user modeling, even though POSGs do not typically explicitly represent it, and how this affords many types of solutions. We also show how different concepts of interaction, such as adaptation, augmentation, delegation, and modulation, follow from this model.
- We illustrate how the interaction model can be used to describe various interactions through two worked-out examples.
- We present an implementation of the interaction model in the form of a library.

We believe this interaction model to be a strong conceptual tool that can be used to facilitate understanding, communicating about, and solving HCI research problems.

2 Related Work

Researchers in HCI have long sought to make humans and computers collaborate efficiently. Initially perceived as a task allocation problem [11, 32], it quickly became one of cooperation. Already in the early 1960's, Licklider described human-computer symbiosis [71], in which computers can devise their own procedures to help humans achieve their goals. Far from being accomplished [108, 109], this vision lives on today in HCI through the field of intelligent user interfaces [72, 81], and through concepts such as human-computer partnerships [96] or human-computer integration [27], and is generally tightly coupled to our definition of interface assistants. Collaboration between humans and artificial systems is also important in fields such as Human-Robot Interaction, where collaboration is mostly physical [23], and artificial intelligence [21], where decentralized cooperation between intelligent actors (humans included)

often requires the demonstration of some form of intelligence. Cooperation (competitive or collaborative) between agents has further been studied in game theory, multi-agent systems, autonomous systems, economy, see *e.g.* [1, 21, 94, 95, 104]. It is therefore impossible to cover all the works that treat about assistance, even in a broad sense; whenever possible, we refer to comprehensive references and do not try to be exhaustive.

In this section, we review the concepts of assistance in HCI, Human-Robot Interaction (HRI) and Machine Learning (ML). While HRI and ML see substantial cross-fertilization on the topic of assistance, HCI remains less integrated.

2.1 Assistance in Human-Computer Interaction

The concept of assistance has existed very early on in HCI [28] with systems that add guidance to the user, external to the main interaction and sometimes even built on top of the main interface [38, 39]. More in line with the modern conception of assistive interfaces is the idea of collaborative systems between humans and computers for which we refer to Fischer [31] and Terveen [113] for an overview that is still relevant today.

Historically, heuristic approaches have been popular in HCI [30, 84], with designers relying on handcrafted rules to create assistance that favors simplicity, usability, transparency, and predictable behavior. With the advent of computational tools, other approaches, such as mixed-initiative systems (MIS), user modeling, and machine-learning, have been used to design assistants. Closest to our work among those modern conceptions are MIS: collaborative systems where the system may take facilitating actions at any appropriately deemed time. These are well described in the literature [49, 50] and high-level principles to guide the design of MIS exist (*e.g.* Horvitz [49] lists 12). Design questions such as *"when should the MIS take a particular action?"* [49] or *"when should the MIS solicit input from the user?"* [33] can usually be answered if the user behavior is very well characterized; for instance, the probability that a user could be made to understand an intervention or the user-perceived cost of a particular intervention have to be known [49]. The answers to these design questions are then, similarly to our decision-theoretic model, typically grounded in expected utility maximization. However, while MIS as described by Horvitz [49] allow for sequential updating of goals and preferences, their underlying decision-making process remains myopic, as MIS compute expected utilities only with respect to their immediate costs and benefits. By contrast, in POSG-based models, actions can have long-term consequences, *e.g.*, with value arising only after a few steps. This enables an assistant derived from a POSG to take actions with delayed benefits, such as gathering information, and to interpret user actions in the same long-term way, *e.g.* recognizing that the user may be performing an action only to *teach* the assistant [42, 102].

Other relevant work include model-based menu design [114], including work in adaptive systems [34, 35], which also feature notions of interaction cost, probability and sequential decision-making in more restricted settings, and model-based design engineering [97], where policies are usually rule-based. Unfortunately, these approaches have not led to a general framework that is widespread in HCI [91].

In modeling users, Norman [87, 88] describes seven stages (goal, plan, specify, perform, perceive, interpret, compare) for how individuals form goals, take actions, and interpret feedback, but without considering assistance. Most of these stages can be interpreted as components of decision-making models such as policies, actions, and inference.

2.2 Assistance in Human-Robot Interaction

The notion of assistance has been also well studied in Human-Robot Interaction (HRI). While overall similar in character to how HCI treats assistance, HRI puts particular emphasis on continuous inputs and physicality, because the assistant is usually a physical robot rather than a software interface. For example, a robotic arm may assist a participant by reaching for and grasping an object, or via combination of user and assistant input [23, 57].

Multiple works in HRI have formalized assisted interaction using decision-theoretic models. For example, Kaupp et al. [64] treat assistance in HRI as a problem of effective communication, basing the robot's decision-making in notions of information gain about user latent goals and intentions. Similarly, Dragan and Srinivasa [23] and Javdani et al. [57] use decision-theoretic models to describe sequential assistance processes in HRI. There, the user is assumed to hold a latent goal state that the robot continuously infers based on user actions, and the robot select actions to minimize the expected interaction costs to reach those states. Recently, these decision-theoretic models have been extended to multi-agent frameworks [42, 102].

2.3 Assistance in Machine Learning

The concept of user assistance is also lively in the field of machine learning [21]. There, assistance is usually considered in two steps: first, inferring users' goal and/or incentives [42, 83, 102], second, forecasting the future and acting appropriately given such beliefs. *Prediction* [21] and *inference* [21, 50] are thus often main drivers of assistance. These inferences and predictions can be model-based or entirely data-driven (see the Introduction in [24] for a list of references), but the latter will usually require normative assumptions [4, 15] to be effective, *e.g.* priors over possible reward functions. Decision-theoretic models are of common use in this context, for example, Fern *et al.* [29] suggest using a POMDP as a generic decision-theoretic model of assistance. While their work targets assistance with daily (physical) activities such as washing hands, the reasons that lead them to consider POMDPs (imperfect observations of the environment, stochasticity, conflicting objectives, specificity to individuals) apply equally well to our context. It should be noted that there is a considerable overlap between the concepts of assistance in machine learning and HRI [102].

Relation to POSG. Many of the existing approaches to assistance referenced above, such as shared autonomy, MIS, Cooperative Inverse Reinforcement Learning (CIRL), and hindsight optimization, are restrictions of the more general POSG. This can be traced to the POSG's representational power: It can capture two-agent interactions with partial observability, latent human state, and diverse objectives, making it a unifying model of assistance. However, as we discuss in subsection 4.4, generically solving a POSG is computationally unfeasible. Each of the above frameworks thus imposes specific structural assumptions, for example, a known human policy

(learned from prior demonstrations [57], or given as a parametric model to tune [42]), a factorization of the state space as in [57], *etc.* These reduce the POSG to a tractable form.

3 Background: Modeling Assistance as a Single-Agent Decision-Making Problem

This section introduces the decision-making vocabulary and notations used in this work, and presents a single-agent decision-making model of assistance that has gained prominence in HCI. Its comparison with our proposed two-agent model is summarized in Table 1.

3.1 Assistance as a Markov Decision Process

The Markov Decision Process (MDP) is a core model of sequential decision-making; applied to the present context, it models the assistant as a sequential reward-maximizing agent. They are extensively used when sequential outcomes are stochastic (random) but influenced by a decision maker (here, the assistant). MDPs represent the surroundings of an agent (the *world*, the *environment*) by assigning values to *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, *e.g.* tied to the efficient completion of an interaction task by the user, or to their satisfaction.

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 assistant has three possible actions to support the user, then $\mathcal{A} = \{1, 2, 3\}$.
(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 the "interface" or "API" of the MDP.
(3) The transition function $T(s', a, s)$ is the probability $\mathbb{P}(s'|s, a)$ of the state transitioning from state s to s' when the agent selects action a. The transition function encodes the internal logic of the world, and can be used to simulate the effect of actions, including the user's reaction to the assistant's actions.
(4) The reward function $R(s', a, s)$ 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 and potentially the cost of action a, *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 s; essentially, it is the agent's "rulebook" which describes its plays. Having the policy be based only on the present state is enough because of the assumed Markov property in MDPs that all transitions only depend on the previous state and the current action ($\mathbb{P}(s_k|s_{k-1}, s_{k-2}, \ldots, s_0, a_{k-1}, a_{k-2}, \ldots, a_0) = \mathbb{P}(s_k|s_{k-1}a_{k-1})$; hence the name Markov decision process). When the agent "plays" K times according to its policy, it will gather on average a cumulative reward value known as the *return* $\mathbb{E}_{\tau\sim\pi}\left[\sum_{k=1}^{K} R(s'_k, a_k, s_k)\right]$, where $\mathbb{E}_{\tau\sim\pi}$ denotes the expected (average) value when the trajectory $\tau = (s_0, a_0, R_0, s_1, \ldots, R_k)$ follows from the agent policy π[1]. We say that the MDP is solved when we find the policy that maximizes the agent's average return. Concretely, such an optimal policy may, for example, maximize a user's text entry rate or selection accuracy, or minimize their task completion time, physical fatigue, or cognitive load.

Solving MDPs has received significant attention, and (approximate) solutions can be found by dynamic and linear programming [63] and (deep) reinforcement learning methods [79, 110].

3.2 Extension to a Partially Observable Markov Decision Process

A Partially Observable Markov Decision Process (POMDP) is an extension of an MDP to the case where the agent cannot perfectly observe the state of the environment. It offers a more realistic account of assistance, wherein the assistant is not all-knowing. Thus, instead of taking decisions based on state values, the assistant has to reason about the state value given current and past observations, with

- Ω the space of possible observations, and
- $O(o|s', a)$, the observation function that describes the probability of the agent observing o considering the world arrived in state s' after having performed action a (*e.g.* a noisy or incomplete version of s').

A POMDP is defined as the tuple $(\mathcal{A}, \mathcal{S}, T, R, \Omega, O)$, which is just the MDP augmented with the observation concepts above. 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. Solving a POMDP is much more difficult than an MDP, because optimal decisions cannot be made uniquely based on the current observation.

3.3 Single-Agent View of Assistance

Both the MDP and its extension POMDP arrive at the same single-agent model of assistance. Consider a two-agent setting, with a user U and an assistant A. Both agents are capable of observing the world state and producing actions, and let us assume they are taking turns. This makes the world state progress from s (start) to $\tilde{s}$ (state after user action) and finally to s' (state after user and assistant actions). Thus, with a_U and a_A the actions of agents, the sequence of actions and states after one turn is

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

If agent A ignores the fact that there is another agent (the user U), it can still learn that selecting a from state s will lead to s'' and associate this choice of action with the experienced reward $R(s'', a, s)$. From A's point of view, U is absorbed in the world; the

[1] It is also 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 of notations, we don't introduce the discount factor explicitly, but it can be considered to belong implicitly to R.

Table 1: Comparison of modeling choices for assistive systems.

	Two-Agent Model (POSG / Dec-POMDP)	Single-Agent POMDP	Heuristics
Model	Explicitly models two agents, each with separate observations, actions, policies, and internal state.	Human behavior treated as part of the environment or as a hidden variable; interaction limited to inference.	Assistant behavior arises from fixed rules or handcrafted approximations.
Representational Power	*Highest.* Captures strategic interaction, teaching, communication, and differing objectives.	*Moderate.* Handles uncertainty but assumes a stationary/known user policy.	*Low.* Usually lacks beliefs or reasoning.
Strengths / Weaknesses	Most expressive but computationally intractable: requires structural simplifications.	More tractable; many off-the-shelf solvers exist [26]. Cannot model strategic adaptation.	Fast, simple, but cannot plan beyond encoded rules.

problem then reduces to a single-agent optimization problem, *i.e.* a (PO)MDP.

4 Modeling Assistance as a Two-Agent Problem

Having established how assistance can be viewed as a single-agent decision-making problem, we now challenge this notion and offer a two-agent generalization that provides a better fit with the characteristics of assisted interaction. Using a multi-agent generalization of POMDP's known as Partially Observable Stochastic Games (POSGs), we formally define assistance, and show how the model can be solved to produce assistant policies. We conclude by grounding the model's key theoretical assumptions about assisted interaction in HCI.

In the classical single-agent (PO)MDP setting, it is assumed that the world reacts the same way (barring stochasticity) every time action a is taken in state s, *i.e.* the transition function $T(s'', a, s)$ is assumed stationary. However, the transition $s \longrightarrow a_U \longrightarrow s'$ 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. This means learning algorithms may fail to converge towards the optimal policy: In essence, the agent has to learn an ever-moving target.

Another difficulty in 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, which, for example, U has learned to anticipate and hedge against. 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.

If, in contrast to the single-agent view, 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. 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, *e.g.* anticipating that because U lacks a crucial piece of information, it will not perform optimally.
(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 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 inversion: $p(\theta|a_U) \propto p(a_U|\theta)p(\theta)$, where $p(\theta)$ is a prior on U's intentions, *i.e.* reason over the most likely intent parameter θ given the observed user action a_U. It can then try to act pro-actively, *e.g.* to resolve ambiguities [73].

By adopting a multi-agent view, A gains the opportunity to adapt to strategic behavior and to explain or predict U's actions. It mitigates the otherwise higher variability in the observed states, since the latent decisions of U would add uncertainty to the already stochastic environment. Ultimately, this promises to improve sample efficiency (learning is faster) and asymptotic performance (final performance is higher), but at a computational cost. A comparison between two-agent models, single-agent models and heuristics is provided in Table 1.

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) [90, 119]. In this work we consider POSGs because they naturally support partial observations and can consider continuous action spaces [65, 66], both of which are common in HCI, contrary to EFGs. For a more detailed comparison between POSG and EFG, see [111, Section 2].

A POSG (see Figure 1 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*.

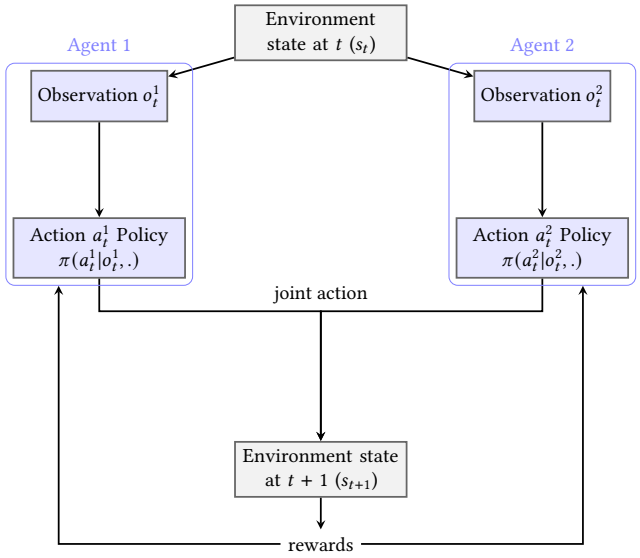


Figure 1: Minimal representation of one cycle of a POSG with two agents. Both agents observe the present state, based on which they take an action. The two actions make the environment transition to a new state, during which it outputs rewards for both agents.

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 $i \in \{1, \ldots, n\}$, 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 $a = (a_1, \ldots, a_n) \in \mathcal{A}$. A joint observation is $o = (o_1, \ldots, o_n) \in (\times O_i)$. T is the state transition and observation probability: $T = \mathbb{P}(s', o|s, a)$, 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. The transition function $T(s', s, a)$ can be separated from the observation function $O(o|s', a)$, as in our exposition of the POMDP, without loss of generality. 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 transition and observation functions that describe 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.

POSGs are a superset of various decision-making models, as illustrated in Table 2. The most basic model here is the MDP, a sequential single-agent decision-making model 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) if all agents are assumed to receive equal rewards. When agent observations are imperfect, the single-agent version of the 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).

Table 2: 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 [51]	MMDP [10]	equal/shared rewards
		Markov games [107]	any reward
Imperfect	POMDP [61]	Dec-POMDP [89]	equal/shared rewards
		POSG [44]	any reward

4.2 Characterization of Assistance

In line with existing definitions in HRI/ML [102], we define assistance in a POSG as behavior of the non-user agent that is intended to maximize user rewards. Formally, the *expected cumulative user reward* for a policy pair (π_U, π_A) is the expected value of the sum of the rewards obtained by the user when the user selects actions according to π_U and the assistant selects actions according to π_A:

$$J_U(\pi_U, \pi_A) = \mathbb{E}_{\tau \sim \pi_U, \pi_A}\left[\sum_k r_{U,k}\right], \tag{2}$$

where notations have been simplified for readability. The point of assistance being to serve a particular user, the best assistance π^* is defined as:

$$\pi^* = \underset{\pi_A}{\operatorname{argmax}} J_U(\pi_U, \pi_A). \tag{3}$$

This formulation naturally lends itself to a Dec-POMDP perspective, where both the user and the assistant act as agents with potentially asymmetric information but a shared objective centered on maximizing the user's cumulative reward. However, the assistant can usually not directly access $r_{U,k}$; even if it could we can still imagine that the assistant may reason differently from the user. For example, the assistant may optimize for long-term cumulative reward, thereby compensating for the user's tendency to prioritize short-term gains [18, 78]. Thus, we adopt the more general POSG perspective which allows for both agents optimizing different reward functions.

4.3 Assistance as a Partially Observable Stochastic Game (POSG)

We now suggest a sequential turn-by-turn model of assistance in line with the common conception of assistance described in Equation 1, and show it is equivalent to a POSG.

We consider the canonical task-user-assistant triplet from Equation 1, and assign each an *internal state*, respectively S_T, S_U and S_A. We further assume a sequential interaction model; agents update their internal state and take actions one after the other, which defines four *turns* as illustrated in Figure 2:

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 four turns define a *round*; the rounds continue until the interaction finishes.

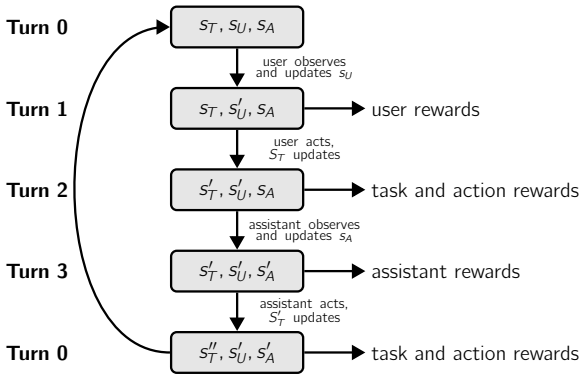


Figure 2: Turns in a round of interaction. A round starts with the user observing the joint state, after which it updates its internal state. The user then selects an action, which makes the task transition to another state. The assistant then follows with the same turns. This repeats until the end of the interaction.

The aforementioned turn-by-turn model is equivalent to a POSG, where the state of the environment is the joint task-user-assistant state. The detailed explanation is delayed to Appendix A, but essentially, it relies on the fact that turn-by-turn models can represent simultaneous decision-making models with proper use of *No-ops* [112]. Since state transitions may be due to external (task) or internal (agents) circumstances, we introduce internal and external transition functions. Finally, 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 which define how agent i produces an observation o_i of the joint state $S = \{S_T, S_U, S_A\}$ through the probability $\mathbb{P}_i(o_i|S)$.

Internal transition functions The internal transition functions, which define how the agent's internal states are updated based on their observations (respectively o_U and o_A) through the probabilities $\mathbb{P}_U(s'_U|o_U, s_U)$ and $\mathbb{P}_A(s'_A|o_A, s_A)$;

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

External transition function The external transition function, which defines how the task is updated based on agent i's action through the probability $\mathbb{P}(s'_T|s_T, a_i)$.

Each of these functions may return a reward (possibly null). The sum of these rewards for each agent constitutes the agent rewards in the POSG. The complete model is illustrated in Figure 3. Compared with the traditional POSG, it makes internal states and their transitions explicit, allows for turn-by-turn interaction, and permits associating rewards with internal state transitions, observations and policies. It thus provides a better fit to the canonical conception of assistance in HCI.

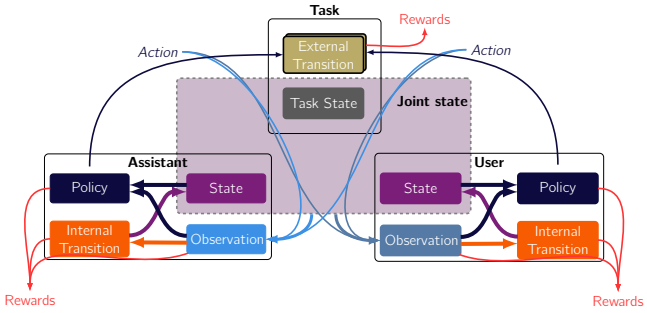


Figure 3: Components of the interaction model. The task and the two agents (user and assistant) are described by a state. Taken together, these form the joint state. Each agent's behavior is further described by 1) the observation function $p_i(o_i|S)$, which specifies how the joint state is perceived 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. These functions are further detailed in Appendix A.

4.4 Solving POSGs by Reduction

Decision-making models can be used purely as descriptive tools. However, given that optimal policies obtained by solving decision-making models can be used to model user behavior or to drive the design of intelligent systems (see subsection 4.5), it is also important in HCI contexts to solve POSGs. In the single-agent decision-making problem, finding a solution amounts to determining the policy of the agent that maximizes 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 each agent are sufficiently close to each other, *i.e.* that the game is not too unequal? One recurrent objective is to find stable (Nash) policies [82], *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 one towards which they may realistically converge. Even when we specify what we mean by solving POSGs, often that solution may be computationally intractable [44]. This is due to the generality of the mathematical framework of POSGs which leads to an exponentially large solution space that cannot be searched efficiently [48]; we return to the computational complexity of POSG in section 8.2.

This does not mean POSGs can never be used outside of a descriptive purpose. Indeed, POSGs will, under some simplifying or restrictive assumptions, often be reducible to subclasses which can become tractable [67]; one example reduction is shown in Table 2. As another illustration, consider the two-agent POSG, with one *user* agent and one *assistant* agent. There (see Table 3), we consider as restrictive assumption the status of the agent's policies *i.e.* whether known or learned, and the impact it has on the POSG and on its potential use in HCI:

- 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. This scenario is often used in HCI to evaluate a system/interface in simulation [80].
- If the assistant policy is known but not the user's, the POSG reduces to one of its single-agent variants, typically a (PO)MDP. Solving the POSG is then equivalent to learning the policy that maximizes the user's reward. This case is often used in HCI to produce user behavior models, *e.g.* touch-typing with an autocorrect [59].
- 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. It is used in HCI to design optimal assistants as in [86].
- 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 has, for example, been considered in HCI to design interfaces that adapt to users who learn over time [41, 68].

Often we cannot know the policy of the agent(s) exactly. Still, many other reductions remain possible. For example, in Cooperative Inverse Reinforcement Learning (CIRL), Hadfield-Menell et al. [42] introduce a two-agent Markov game, but show it can be reduced to a (single-agent) POMDP with latent (unobservable) user states. Similarly, in their bounded memory model of mutual adaptation, Nikolaidis et al. [85] 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 subclasses of POSG exist which make solving POSGs computationally less intensive, for example, regarding observations (public observations [37, 48], *i.e.* whether agents share some of their observations; one sided-observations [17], *i.e.* one agent has partial and the other has perfect observations), or actions (public [37], *i.e.* the actions of the agents are visible to each other). POSGs thus offer a powerful descriptive tool to capture the mechanisms of assisted interaction while, at the same time, allowing for more practical and tractable reductions to model user behavior or identify effective assistant policies.

Table 3: 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 Key Theoretical Assumptions

Applying POSG to model assistance implies strong theoretical assumptions about interaction with assistive interfaces. We discuss each one and comment on their alignment with the HCI context.

User characteristics are hidden states. Different users possess different characteristics and abilities [118]. We assume that these can be represented and quantified by a value; for example, user expertise could be represented by a number between one and ten, 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 [50].

Future states only depend on the present state. The decision-making models presented before all make the hypothesis of Markovian state sequences, *i.e.* that state transitions only depend on the last visited state and (joint) action, the last state encapsulating all relevant historical information: $\mathbb{P}(s_{k+1}|s_k, \ldots, s_0, a_k) = \mathbb{P}(s_{k+1}|s_k, a_k)$. It is however common in HCI to have states dependencies that last more than one state transition. For example, a memory model may be considered that requires information from several past seen items [116]. Although sometimes it can be deduced how far back the state dependencies extend, it can also be empirically tested by considering a null hypothesis $H_0 : s_{t+1} \perp s_{t-k}|s_t, a_t$, which asserts that the state visited k steps ago provides no additional information about the next state transition beyond what is already known from the current state.

If indeed dependencies extends to several states, one may augment the state with past states (frame stacking in [79]), which recovers the Markov property by construction. 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. This does come

at the cost of increasing the dimension of the space, which makes the decision-making problem harder to solve. Another solution is to introduce latent states in the state space that summarize the human's memory and internal state; this allows history-dependent human behavior to be treated as if it were Markovian. These latent states can be learned implicitly (*e.g.* as in Dreamer [43]; see state representational learning [25]), or they may arise from explicit cognitive modeling of the user as in [86]. In HCI, the latter approach has been the most popular.

Observations are incomplete. User and assistant observations are often incomplete and/or noisy: Many user states, being psychological constructs, are latent, and the assistant's internal states are often hidden from the user. Although there are ways to explicitly signal assistant's state to the user, as in Octopocus [5], where feedback about the perceived likelihood of each gesture is signaled in real-time to the user. Höök [47] also discuss how to create "glass boxes" based on formalism that signal their understanding to the user. Since users also have limited perceptual and cognitive abilities (*e.g.* imperfect vision, decaying memory), states that are perfectly observable may cease to be depending on the considered time scale. Similarly, assistive interfaces may collect noisy and incomplete sensor measurements.

User behavior is reward-driven. Solution concepts for decision-making models often revolve around *rational* agents that maximize cumulative rewards. The assumption of the rational user which maximizes its utility is an old one, even appearing before Morgenstern and Von Neumann's work in economy and game theory, and has been challenged multiple times [62]. Still, it can be an appropriate framework for modeling users, as one can always choose to model users as rational, regardless of whether the user is truly rational; the question then becomes which reward function makes the observed behavior optimal, which forms the basis of inverse reinforcement learning (IRL). In IRL, 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 that it is unique [58]. Thus, the rationality assumption does not necessarily reflect a normative assumption, which has been the recipient of justified criticism; it may just be a generic, unfalsifiable way to specify the objective of interaction as the accomplishment of goals [45].[2] In practice, 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 [19, 52, 53, 59, 105], and user preferences [101] and goals have also been conveyed via rewards.

Designing by maximizing scalar rewards. Optimizing an objective cost function has shown to be a successful design strategy for interfaces [92]. 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 which, when optimized, returns the "best" design. This can be operationalized in decision-making models by setting appropriate 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, *e.g.* relating to comfort or ergonomics [18].

[2]To recover normative force, additional assumptions must be imposed to the reward functions. This is precisely the motivation for bounded rationality frameworks [70], which posit that human (user) behavior emerges from free choices the user makes in their best interest, within constraints imposed by the environment.

5 Understanding Assistance across HCI

We characterized assistance as resulting from maximizing the expected cumulative user reward in a two-agent game (see subsection 4.2). We then described several methods for reducing POSGs in different ways to provide assistive policies (*e.g.* CIRL [42]). Now, we discuss how the decision-making model affords formally defining *types* of assistance commonly discussed in HCI, *i.e.* different interaction paradigms that place further restrictions on the POSG. These illustrate the model's representational power to capture meaningful concepts in assisted interaction. The types are summarized in Figure 4; they are not mutually exclusive, nor are the examples exhaustive.

5.1 Adaptation

Adaptation has been examined prominently in the study of adaptive interfaces, usually defined as systems that modify aspects of their behavior or presentation during use [54]. Such adaptive mechanisms appear in contemporary systems, *e.g.* Todi *et al.* 's adaptive linear menu [114], which adjusts item order based on predicted relevance and selection speed. What qualifies as an adaptive system remains contested: Most definitions agree that adaptation involves on-line changes to system behavior in response to usage [117]. However, some scholars argue that a system is adaptive only when such changes are based on explicit or implicit user models [12, 46, 55]. As a result, classifying specific systems can be ambiguous. For instance, the Leitner system described later in section 6 is sometimes labeled adaptive [86], yet it is a purely rule-based algorithm. Our model has the representational power to define adaptation.

Let the assistant policy at time step k be defined as $\pi_{A,k} = \pi_{A,k}(h_1^k)$, *i.e.* the assistant policy is a function of the entire history of the assistant observations $h_1^k = \{o_{A,1}, o_{A,2}, \ldots, o_{A,k}\}$. We say that a system is adaptive if the assistant's policy changes over time as a function of past interactions: $\exists k$ such that $\pi_{A,k+1} \neq \pi_{A,k}$.

When a system with a dynamic assistant policy adapts to a user that also changes policy in response, we talk about *reciprocal adaptation* [41]. Importantly, parameters (but not states) that directly affect action selection, such a gains or weights, are considered part of a policy. For example, a neural network whose structure does not change over time, but sees its weight updated, is still considered an adaptive policy.

A further advantage of our definition of adaptation being based on a decision-making model is that it can be quantified:

- Adaptation magnitude can be measured by a well-chosen distance between two policies *e.g.* a KL divergence. This can be useful to determine by how much a policy changes.
- Adaptation rate is the frequency of non-zero adaptations. This can be useful to determine how often a policy changes, *e.g.* if the changes are after each user input or not.

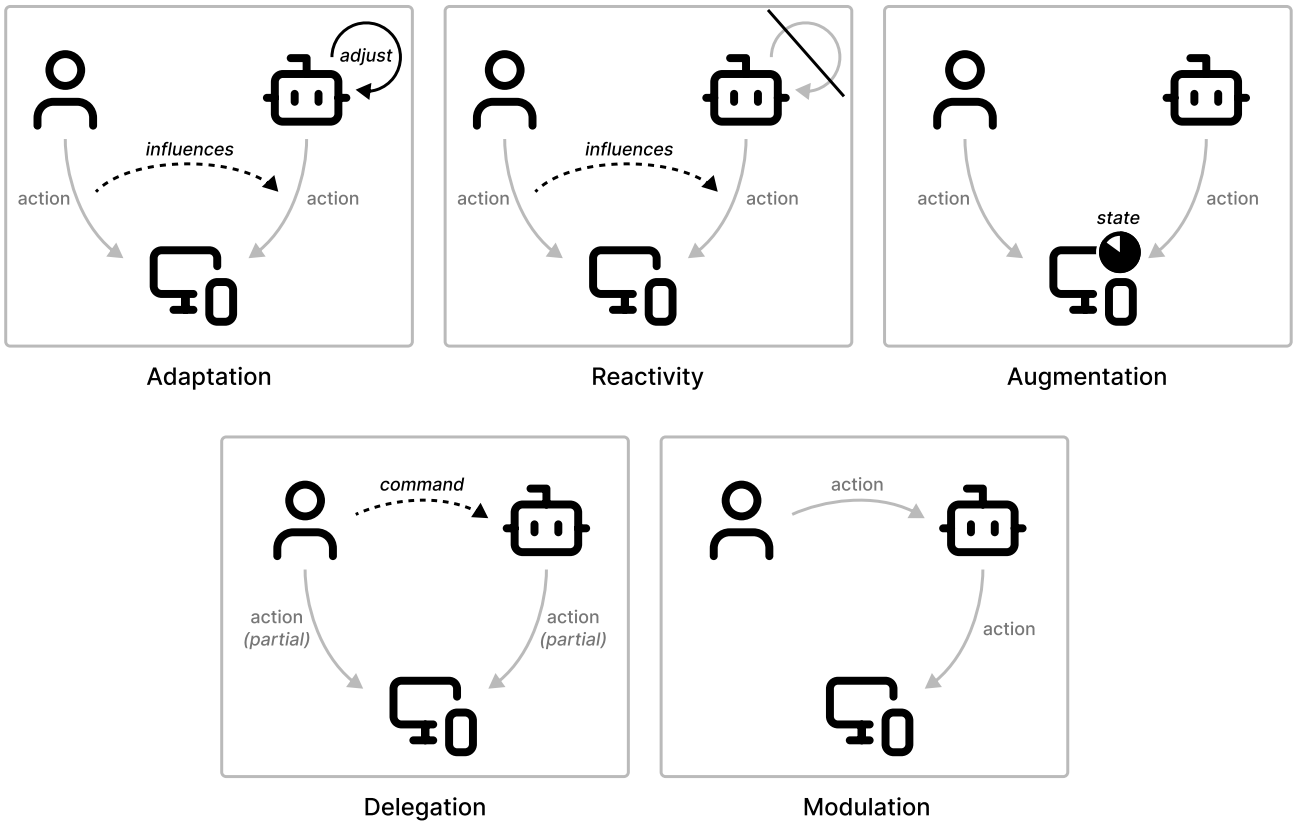


Figure 4: Different types of assistance. In *adaptation*, the assistant changes its policy in response to observed user behavior. In *reactivity*, the assistant's policy is static, but is a function of (past) user actions. In *augmentation*, the assistant actions affect states that are otherwise inaccessible to the user. In *delegation*, some tasks can be performed by both agents. In *modulation*, user action are necessarily mediated by the assistant.

- Adaptation horizon is the size of the effective time window considered by the adaptive policy. This allows estimating the effective memory of the policy, and as a result, how fast it can react to new user behavior.

5.2 Reactivity

When the assistant policy is static, and only responds to observations according to a pre-programmed "playbook" (policy), we talk of a reactive system $\forall k, \pi_{A,k+1} = \pi_{A,k}$. Reactive systems can further be separated into two classes:

- *immediately reactive systems*, where the policy only depends on the last observation $\pi_{A,k} = \pi_{A,k}(o_{A,k})$, and
- *history-dependent reactive systems*, where the policy depends on at least two past observations $\pi_{A,k} = \pi_{A,k}(h_i^k), \quad i < k$. This includes stateful systems, which update their internal states based on each new observation: $\pi_{A,k} = \pi_{A,k}(S_A)$ and $S_A = f(h_i^k), \quad i < k$.

A neural network that has been trained offline on data, and is never updated on-line is thus considered reactive, even when its input might be past or present user actions. It is not adaptive, because it remains a fixed set of rules, even though they have been learned and not hard-coded by a designer. Examples of reactive assistance can, for example, be found in view and layout management systems for mixed reality, such as AUIT [6] or SemanticAdapt [20].

5.3 Augmentation

An assistant which performs *augmentations* allows the task to transition to states that the user cannot reach. In other words, augmentation is when there exists at least one task state value s_ϕ that can be reached only via the assistant external transition function. Formally:

$$\begin{aligned} &\exists (s_\phi \in \mathcal{S}_T, s \in \mathcal{S}_T, a_A \in \mathcal{A}_A) \text{ such that:} \\ &\quad p_A(s_\phi | s, a_A) > 0, \text{ and} \\ &\quad \forall a_u \in \mathcal{A}_u, s \in \mathcal{S}_T, p_u(s_\phi | s, a_u) = 0 \end{aligned} \tag{4}$$

In intelligent text entry systems, suggested completions displayed on top of the main keys are augmentations.

The amount of augmentation provided by the assistant can be computed based on comparing the set of augmented states Φ to the set of task states $\|\Phi\|/\|\mathcal{S}_T\|$, where $\|\cdot\|$ is an appropriate measure, *e.g.* the cardinality of a set.

5.4 Delegation

In *delegation*, there are some tasks that can be performed by both the user and the assistant, *i.e.* some state transitions are achievable both by the user and the assistant. Formally:

$$\exists (s', s) \in \mathcal{S}_T^2, (a_U, a_A) \in \mathcal{A}_U \times \mathcal{A}_A, \text{ such that:} \\ p_U(s'|s, a_U) > 0 \text{ and } p_A(s'|s, a_A) > 0. \tag{5}$$

The amount of delegation can be computed by counting the number of times the state transitioned from s to s' thanks to the user or assistant actions. Let $N_U(s'|s, a_U)$ and $N_A(s'|s, a_A)$ be those respective numbers. Then for a particular pair of states (s', s), we can compute the delegation fraction as

$$D(s', s) = \frac{N_A(s'|s, a_U)}{N_A(s'|s, a_U) + N_U(s'|s, a_U)} \tag{6}$$

This fraction is bounded up by 1 (full delegation) and down by 0 (no delegation). An average delegation score can be defined as

$$\mathcal{D} = \mathbb{E}[D(s', s)] \tag{7}$$

The value of $\mathcal{D}$ can be seen as a way to quantifies a system's *autonomy*: The more tasks are being delegated to the assistant, the more autonomous it is. An example of a system with delegation is a collaborative classification tool, where both the user and the assistant can classify objects (*e.g.* [60]).

5.5 Modulation

In *modulation*, user actions cannot directly influence the task state; instead all user actions are mediated by the system:

$$p_u(s'|s) = \begin{cases} 1 & \text{if } s' = s \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

and the task state evolves only through assistant actions. An example are mouse transfer functions [13], and more generally, most interaction techniques, which translate user behavior, such as physical actions, into system-interpretable actions.

6 Examples

In this section, we present two illustrative examples of how our model can be applied to model assistance. The first one is a pointing facilitation technique. We chose this example because most researchers would not treat it as a multi-agent problem. The second one is an intelligent tutoring system, which more consensually could be treated as a multi-agent problem. For both illustrations, we pair the assistant with a synthetic user model rather than a real user, to allow us to reason about user states. We finally discuss how the two examples illustrate the descriptive power and flexibility of our decision-theoretic model. These examples are worked-out in the accompanying library described in the next section. In general, the process to apply our model to new assistive problems follows these steps:

(1) Identify the task and the assistant components of the interface.
(2) Define the state variables, that is, the quantities relevant for decision-making. These describe both the task and the agent's internal states (*e.g.* beliefs, goals, or cognitive states). Note that increasing the dimensionality of the state space makes solving the model more difficult. Clearly specify each state variable and its domain.
(3) Specify the actions available to each agent.
(4) Define the observation model, describing which aspects of the state each agent can observe. This includes task states and any observable internal states. Typically, the internal state of the other agent is not directly observable.
(5) Define the external transition function, which describes how the task state evolves given agents' actions.
(6) Define the internal transition functions, which specify how each agent's internal state updates after an observation.
(7) Specify the agents' policies, when the goal is not learning them. The policy is any mapping from observations to actions.
(8) Define the reward functions, encoding the (assumed) objective of the user and of the assistant. These determine the preferred behaviors at each step, and are used evaluate or train optimal policies.

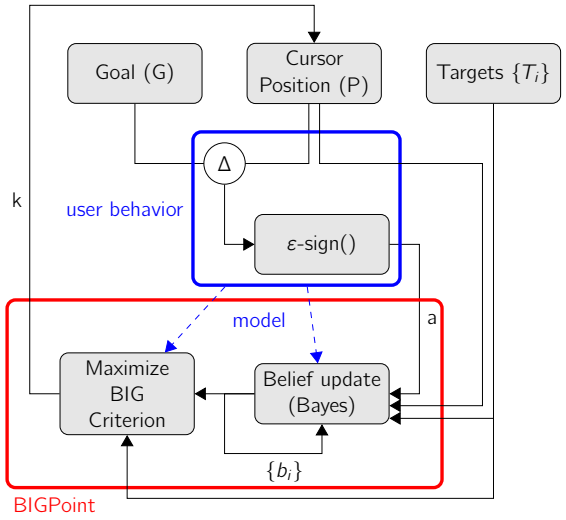


Figure 5: The pointing facilitation problem. Pointing facilitation is modeled as a 1-dimensional navigation problem. If the goal G is right of the current cursor position P (*i.e.* $G > P$), the user selects action $a = 1$ (move right) with probability $1 - \varepsilon$, else $a = -1$ (move left) with 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 to determine the next cursor position by maximizing the BIG criterion. Blue indicates user components, while red indicates assistant components.

6.1 Example 1: A multi-agent decision-making model of pointing facilitation

Pointing facilitation techniques assist users in moving a virtual cursor onto an intended target. They map physical input, *e.g.* physical movement speed represented as "mickey counts", to the movement of a virtual cursor as output, usually in pixels, with the aim of improving the accuracy and speed of target acquisition. Pointing facilitation techniques have received wide attention in HCI research

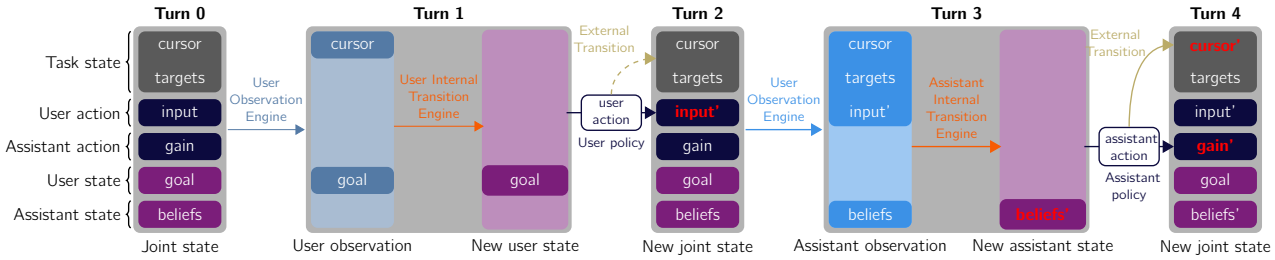


Figure 6: A round of interaction in the pointing facilitation example. The game state for any given round (turn 0) is displayed left. During the first turn, the user produces a partial observation of that state; its internal state remains unchanged, since the user's goal does not change during the task. 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, while arrows indicate inputs and outputs of the various components of the user assistance model. The colors map to those of Figure 3.

(*e.g.* [8, 9, 13, 14, 40, 69]). Here, we describe BIGPoint, a variation of BIGNav [73] without zoom. BIGPoint is a belief-based pointing facilitation technique where the assistant maintains a belief about the user's intended goal. The pointing facilitation problem and BIGPoint's underlying mechanism are illustrated in Figure 5.

6.1.1 *Components of the Assistance Model.*

- The *task state* consists of the n potential targets T_i and the cursor position P: $S_T = \{P, \{T_i\}_{i=1}^n\}$.
- The *user's state* contains the goal target $S_U = \{G\}$.
- The *transition function* describes how the cursor moves according to a CD-gain [13, 14]: $P_{j+1} = P_j + a_U \times a_A$
- *User actions* are {left, right} represented as $a_U \in \mathcal{A} = \{-1, 1\}$. Thus $a_U = \text{sign}(\text{Goal position } (G) - \text{Cursor position } (P))$. We assume as user model one which makes mistakes with probability ε, thus $\pi_U(a_U|S_T, S_U) = \pm\text{sign}(P - G)$ with probabilities ε and $1 - \varepsilon$.
- The assistant selects the CD-ratio a_A.
- The goal of the designer is to minimize the number of user actions, thus the *reward* given is -1 on each round.
- The *assistant* maintains a belief *state* $\{\{p(T_i)\}_{i=1}^n\}$ over all targets T_i, obtained by Bayesian updating, with the likelihood being a model of the user's policy.
- Each agent is able to perfectly *observe* the task state and their own, but not the others.

With these elements in place, one can now try to find the optimal assistant policy. This could be a heuristic policy fixed by trial and error, one learned by reinforcement learning, or any other method that the researcher chooses. In BIGPoint, the assistant policy is recomputed after each user action to maximize a criterion proper to BIG [73, 74] that best reduces the expected uncertainty about the user's goal target. A round of interaction using the turn-by-turn model is schematized in Figure 6.

6.1.2 *Classic pointing facilitation techniques as different policies.*

The above description generalizes to other techniques: By assuming different forms of assistant policies, we retrieve many existing pointing facilitation techniques from the HCI literature, which shows the modularity and power of the representation.

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 [13, 14].

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 [13, 14]. 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 [69].

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 [40] and semantic pointing [8] are two well-known examples (also see [56, 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, compared to target-aware techniques, further reasons about the user goal.

6.2 Example 2: A multi-agent decision-making model of artificial teaching

In the above example, we looked at a low-level form of modulation that helped the user to efficiently move a mouse cursor to a target item. In this example, we consider a higher-level form of assistance. We revisit the model by Nioche *et al.* [86] on artificial teachers for learning foreign languages within our multi-agent decision-making model. Originally, an assistant proposes an item among N possible (a word in a foreign language) and the user has to respond with the equivalent word in their 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 learned 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 learned 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 Figure 7.

6.2.1 Components of the Assistance Model.

- 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 (whether the user recalls the presented item or not). The recall probability associated with each item is determined by a memory model (in Nioche *et al.* 's work, 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 in Figure 8.

6.2.2 Different teaching strategies as different policies. The artificial teaching description captures 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, it can have reasonable performance if the user's recall is good, since this policy ends up presenting many items.

Spaced repetition With spaced-repetition systems, the assistant policy is a rule-based state machine system that takes as input the last user response, *e.g.* the Leitner system [86, 99].

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 [86].

The two illustrative examples highlight several benefits of our proposed interaction model: While the pointing facilitation and artificial teaching examples are very different problems at different levels of assistance, we represent them without ambiguity with the same elementary mathematical objects, 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 the assistant policy. The examples thus demonstrate the framework's ability

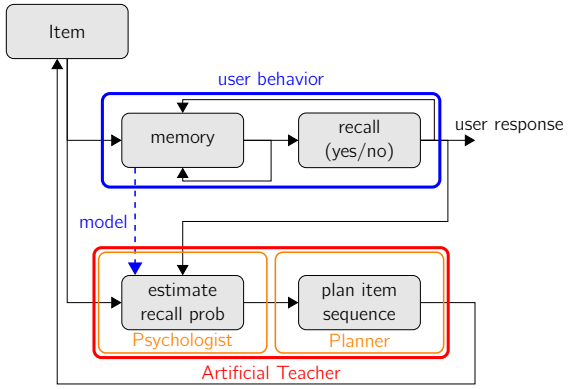


Figure 7: The artificial teaching problem, and Nioche *et al.* 's [86] solution. When an item is presented, the memory of the user is solicited, and the user recalls the item or not. The artificial teacher has a *psychologist* module that estimates the user's recall probabilities at each time using a memory model. It also has a *planner* module, which constructs the sequence of items to be presented based on the psychologist's output.

to capture and describe various interaction patterns. At the same time, it does not prescribe optimal interventions, and leaves the researcher full freedom when it comes to specifying the assistant policy. These could be obtained by finding an optimal policy, *e.g.* by solving the decision-making model, but not necessarily. They may also include, for example, hand-crafted assistant policies, based on heuristics.

7 Implementation: the CoopIHC Library

In the previous section, we illustrated the descriptive power of our decision-theoretic model to capture various assistance forms and strategies. To demonstrate that the decision-making perspective is not only theoretically appealing but also practically actionable, we developed a software library called CoopIHC that instantiates the proposed turn-by-turn model of interaction. As decision-making models are defined in terms of unambiguous mathematical objects (states, actions, observations, and rewards), they naturally lend themselves to implementation. As discussed in section 3, the model itself essentially defines an application programming interface (API). As a library used to specify decision-making models, CoopIHC sits in between libraries destined to solve decision-making models (*e.g.* RLLib, Stable-Baselines, quickpomdps, PyGAD) and domain-specific user modeling libraries (*e.g.* MuJoCo for biomechanical simulation, pyACTR for cognitive modeling, *etc.*).

Design philosophy. The design principle is to map theoretical entities of the turn-by-turn model directly to programming constructs.

- User and assistant objects can be "bundled" together with a task object. This provides a modular definition of the "game".

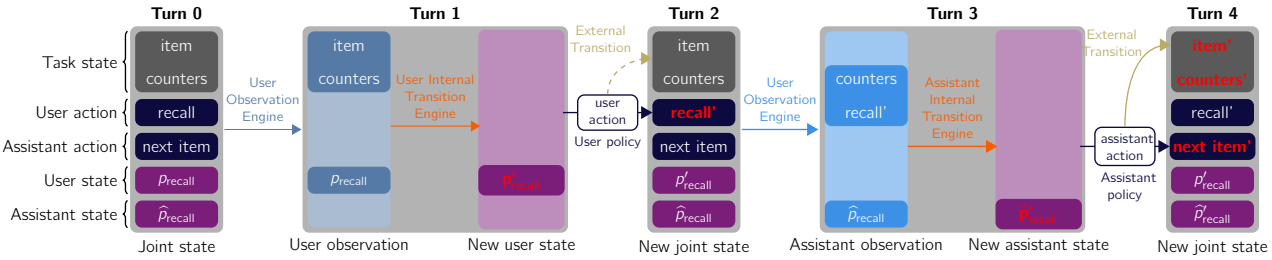


Figure 8: A round of interaction in the artificial teaching example. During the first turn, the user produces a partial observation of the game state, and the recall probabilities are updated (memorization). The user recalls the item or not, and the state is updated (turn 2). In the third turn, the assistant 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.

- Each of these objects is described by a state, which takes value in a well-defined space. Agents also have actions. The joint state is the reunion of all three states and agent actions.
- Observations engines are objects that map the joint state to observations. Each agent has its own observation engine.
- Inference engines are objects that take the agent's internal states and observations as input, and output an updated internal state. Each agent has its own inference engine.
- Policies are objects that take agent internal states and observations as input, and output an action. Each agent has its own policy.
- Observation and inference engines, and policies all return rewards whenever their main method is called.
- The "Bundle" object orchestrates all objects in line with the turn-by-turn interaction model.

The correspondence between components of the turn-by-turn model and CoopIHC objects is detailed Table 4. Each of these components is a first-class object; defining a new component consists of subclassing a base component and overriding one (or several) methods. This ensures a common interface between components and reduces the amount of boilerplate code. It also ensures that the library is agnostic to any one solution, since virtually any piece of code can be defined inside a method. For example, a policy may be effectively defined by a simple rule, an algorithm, a neural network, an LLM, and even an object like an entire *Bundle*. Code snippets can be found in the library's documentation.

Features of the library. A full technical presentation of the library is beyond the scope of this paper. Nevertheless, the implementation is supported by extensive documentation and automated tests. The library code and documentation is currently hosted at https://jgori-ouistiti.github.io/CoopIHC/.

8 Discussion

In this paper, we have proposed decision-theoretic modeling as a unified and actionable lens to assistive interfaces. We have modeled assistance as a turn-based interaction between two agents, a user and an assistant, that operate under uncertainty about the dynamics of the interaction task and about the other agent, aiming to maximize a reward signal benefiting the user.

8.1 Benefits of the Multi-Agent Decision-Making Perspective

Two-agent view for the modeler, single-agent view for the agent. First, we have argued that a two-agent formulation offers a significant advantage over the single-agent view. 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). However, from the assistant's perspective, *i.e.* at *execution/deployment time*, the entire system can be reduced to a single-agent problem. This reduction does not invalidate the two-agent view; in fact, it is necessary to make the assistance problem computationally tractable. Given the currently available algorithms to solve multi-agent decision-making problems, an assistant will most likely operate under a single-agent model. However, assistance derived from a two-agent model is very different from an assistant policy that originates from a single-agent model assuming a fixed and passive environment. In the two-agent case, the derived single-agent model still encodes human behavior in some shape or form. Concretely, in the single-agent case, the human is treated as noise/variability, whereas under the two-agent model, it is treated as the source of latent variables, meaning some of the variability can be "modeled away".

Clarity about modeling assumptions. Second, assumptions about the assistant, its task, and the user are made explicit under the multi-agent view. Using our decision-theoretic model requires specifying what the assistant observes, what it does not observe, what the user perceives, and how both agents' goals are defined. In this way, the model leaves little space for hidden assumptions and provides a single "language" for comparing alternative designs. This is valuable especially when comparing competing approaches. For example, the model can help clarify what is meant exactly by notions like "adaptation" or "context-awareness". This can be seen in the two worked-out examples, in which various solution approaches regarding pointing facilitation and teaching assistance can be mapped

Table 4: Correspondence between CoopIHC and the assistance model. Each entry in the table gives the correspondence between the components of the user assistance model and CoopIHC classes that the end-user subclasses to make their own objects. Object-specific methods that the end-user overrides to determine the object's behavior are also provided, which shows how the library fits the model. Starred classes (*) are not subclassed but directly instantiated.

Name in our Model	CoopIHC Base-class	Specific Methods to override	Comments
Agent	*BaseAgent*		collection of state, observation, internal transition and policy
Task	*InteractionTask*	*on_user_action* *on_assistant_action*	external transition function (user) external transition function (assistant)
Observation function	BaseObservationEngine	*observe*	performs the observation
Internal transition function	*BaseTransitionEngine*	*transition*	updates the internal state
Policy	*BasePolicy*	*sample*	select an action
States	*StateElement**		value of a state's component
	*Space**		domain in which the StateElement takes value
	*State**		collection of *State*s or *StateElement*s

to the same general problem definition, as well as in the formal definitions of assistance types in section 5.

Explaining the intrinsic difficulty of assistance. Third, the theory explains why and where assistive interfaces are hard to implement. As the two-agent view shows, many difficulties in assisting users are not simply the result of poorly designed assistants; rather, they reflect structural difficulties posed by the interaction itself. Partial observability, noisy and limited communication, and changing goals all make coordination hard. Beyond clarifying these difficulties, the framework can help rank the difficulty of various HCI problems by looking at the computational complexity of the associated reduced decision-making model. If one can reasonably argue that, say, adaptive menu navigation can be reduced to a POMDP, while collaborative writing with a proactive assistant is better modeled as a Dec-POMDP, then the latter is inherently "harder" in a computational sense. This also shows how it can hamper unrealistic design expectations: If the interaction turns out to be NEXP-hard, we should likely not expect tractable exact solutions; heuristics or approximate solutions would likely be the only way forward. Note that computational complexity is not the only dimension of difficulty in assistance problems, and only characterizes the difficulty of finding an optimal assistant policy for a given decision-making problem. Acceptability, learnability, *etc.* should also be taken into account.

Reusing and incrementally advancing prior work. Fourth, we have shown that the theory is not just a theory (paperware) but can be implemented in code to drive the design of assistants in real applications. As our CoopIHC library and its corresponding implementation of the worked-out examples demonstrate, the decision-making model can serve as a standard for re-using and incrementally advancing prior work. This is necessary for two reasons. First, the need for baselines and benchmarks is acknowledged in most computing fields and is required for high-confidence experiments. Benchmarks that are not exceedingly simple have to be uniquely defined rather than approximately replicated by each researcher, which requires some form of standard. Second, reusing prior work and extending upon it is one primordial mechanism of science. Yet, currently, this is often difficult in HCI, owing in part to the variety of models used to describe systems. The proposed model can serve as such a standard, *e.g.* by means of the CoopIHC library.

Unification across and beyond HCI domains. Finally, we have demonstrated how decision theory can shed light on research on assistive interfaces, which has been previously fragmented across disciplines and systems. What appears at first glance to be a set of unrelated problems can thus be recognized as instances of the same problem of cooperative decision-making between a user and an assistant under uncertainty. This promises cross-domain transfer of methods and insights, and positions HCI problems within other fields, such as multi-agent systems, reinforcement learning, economics, and cognitive science. It thus offers a common language that can facilitate cross-disciplinary exchange: HCI researchers could more easily import methods and theoretical results from these fields, for example, effective solutions to the two-agent problem such as CIRL [42], while also contributing novel case studies that expose new classes of coordination problems.

8.2 Limitations of the Proposed Decision-Making Model

Validity of the proposed definitions. We do not claim that the definitions proposed in this work (*e.g.* those in section 5) are "correct" or universal. A definition is rarely meaningful on its own; it only becomes useful when linked to a property or operational purpose. In the context of this paper, our aim is not to argue for the primacy of any particular definition, but rather to demonstrate how decision-making models provide a systematic framework to define concepts in a precise and operationalizable manner. It is anticipated that certain definitions offered here may require clarification or redefinition, and that additional definitions may need to be formulated.

Are the descriptions really unambiguous? Language is not unambiguous, and mathematics may not be either. Dix [22] argues: *"However, mathematics is ambiguous precisely because it is founded on abstraction. The nature of abstraction is to ignore detail, to focus on one aspect of the world at the expense of others. So it trades total precision about some aspects at the cost of utter ambiguity of others. This is no bad thing—it is the power and strength of mathematics—but it is a wise thing to bear in mind when you use it."* When casting an interaction as a decision-making model, one has to make explicit choices about what the state space is, what constitutes user and assistant actions, *etc.* It is possible that two researchers modeling the same interaction do not end up with exactly the same model. For

example, as explained before, imperfect memory can be modeled both as the result of an internal state transition or as an imperfect observation. Still, the goal is to produce a description that is understood by everyone, even if it may not be unique.

Computational complexity of solving POSGs. Finite-horizon Dec-POMDPs, a special case of POSGs, are NEXP-complete [7]. Therefore, POSGs are at least NEXP-hard, *i.e.* determining that a policy is optimal requires searching over the entire policy space, which grows double-exponentially in policy horizon length [2]. As a result, even small discrete POSGs can often not be exactly solved. Thus, real-world POSGs have to be handled through approximations (*e.g.* Deep MARL algorithms, such as MAPPO, QMIX [2]) or structural simplifications (*e.g.* restrictions on possible policies). A common strategy is centralized training, decentralized execution (CTDE) [2]: During training, a single "centralized agent" observes all agents' observations and outputs joint actions. This shared information means the agents do not need to reason about the private histories of the other agents. This reduces the problem's computational complexity, casting it to a form very similar to a POMDP. Similarly, single-agent POMDPs, although less computationally intensive than POSGs, still cannot be solved exactly in many cases; in practice, all successful approaches in HCI to handle single-agent POMDPs have relied on computing approximations to our knowledge. Future research should identify subclasses of POSG that are meaningful to HCI, yet can be solved efficiently. For example, a common form of decision-making model in HCI seems to be a DEC-POMDP with public actions and factored state and observation spaces. This simplified form may mitigate computational challenges of POSGs, while enabling a unified decision-making framework across different domains.

Strict assumptions of the decision-making models. Our model only accounts for the interaction between a single user and a single assistant performing a task that is not open-ended. The commitments of our model do not show a significant difference between a user and an assistant: users and artificial agents are characterized in our theory by their common ability to observe and act within the world, and by their incentive to make decisions to maximize rewards. This symmetry is in line with a proposition by Gershman *et al.* [36], but may be debated. Another risk is the use of scalar reward functions, which are known to be "gameable" by agents; to mitigate these risks, we refer to [93] for an overview. Further, the "interfaces" of the decision-making models presented in this work are static: user and assistant action spaces are fixed once and for all. This is a limitation, particularly if the goal is to model very dynamic situations such as a user learning new commands. In some cases, we can assume a large action space from the beginning to handle this limitation. For example, in a command selection task, the user's action space may be the set of all possible commands, even those that are yet unknown, as long as there is a mechanism in place that prevents the user from selecting unknown commands. It should also be noted that some works exist on this topic, *e.g.* [16] for MDPs. All of these assumptions imply that the model is better suited to describe well-defined tasks, rather than more open-ended interactions; although LLMs, used to produce observations, actions or rewards of controlled dimensions from complex environments [3, 76], are starting to address this issue. Thus, the ability to solve POSG models does not necessarily depend on the application domain, but rather on how well the problem can be formalized by the two-agent decision-making model, and which of its reductions remain plausible. This becomes increasingly difficult the more factors (cognitive, emotional, *etc.*) affect user behavior, and the more high level the task, which, *e.g.* leaves more room for hierarchies and interdependencies between states, and remains the chief challenge for expanding the applications of this theory.

9 Conclusion

At the beginning of this paper, we observed that the landscape of assistive interface research is fragmented across domains and disciplines, resulting in a field rich in innovation but lacking a common language to connect its diverse approaches. In this paper, we offer one such common language: a decision-theoretic model that frames assistance as a cooperative interaction between a user and an assistant agent operating under uncertainty within a shared task environment. We demonstrated how this model provides a structured vocabulary for making modeling assumptions explicit, a computational basis for comparing disparate assistance strategies, and a means to define common notions in assistance and to explain their intrinsic difficulty. Through our worked-out examples and their implementation in the CoopIHC library, we have shown how this decision-theoretic perspective provides practical utility, offering a reusable foundation for building upon prior work. We believe that these promising results warrant more attention on decision theory as a principled yet actionable approach to assistive interfaces.

Acknowledgments

We thank our colleagues for fruitful discussions, and especially Andrew Howes for comments on an earlier version of the paper. This work was partially supported by ANR-24-CE33-7994-01, ERC Advanced Grant 101141916, and Research Council of Finland (FCAI (328400, 345604, 341763); Subjective Functions (357578))

References

[1] Stefano V Albrecht and Peter Stone. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence* 258 (2018), 66–95.

[2] Christopher Amato. 2024. An introduction to centralized training for decentralized execution in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2409.03052* (2024).

[3] Anonymous. 2025. Designing Observation and Action Models for Efficient Reinforcement Learning with LLMs. In *Submitted to The Fourteenth International Conference on Learning Representations.* https://openreview.net/forum?id=v1tTJN2svr under review.

[4] Stuart Armstrong and Sören Mindermann. 2018. Occam's razor is insufficient to infer the preferences of irrational agents. *Advances in neural information processing systems* 31 (2018).

[5] Olivier Bau and Wendy E Mackay. 2008. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. 37–46.

[6] João Belo, Matthias N. Lystbæk, Anna Maria Feit, Ken Pfeuffer, Peter Kán, Antti Oulasvirta, and Kaj Grønbæk. 2022. AUIT - the Adaptive User Interfaces Toolkit for Designing XR Applications. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22).* Association for Computing Machinery, New York, NY, USA, 16. doi:10.1145/3526113.3545651

[7] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research* 27, 4 (2002), 819–840.

[8] Renaud Blanch, Yves Guiard, and Michel Beaudouin-Lafon. 2004. Semantic pointing: improving target acquisition with control-display ratio adaptation. In

Proceedings of the SIGCHI conference on Human factors in computing systems. 519–526.

[9] Renaud Blanch and Michael Ortega. 2011. Benchmarking pointing techniques with distractors: adding a density factor to Fitts' pointing paradigm. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1629–1638.

[10] Craig Boutilier. 1996. Planning, learning and coordination in multiagent decision processes. In *TARK*, Vol. 96. Citeseer, 195–210.

[11] Jeffrey M Bradshaw, Paul J Feltovich, and Matthew Johnson. 2017. Human–agent interaction. In *The handbook of human-machine interaction*. CRC Press, 283–300.

[12] Peter Brusilovsky. 2001. Adaptive hypermedia. *User modeling and user-adapted interaction* 11 (2001), 87–110.

[13] Géry Casiez and Nicolas Roussel. 2011. No more bricolage! Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 603–614.

[14] Géry Casiez, Daniel Vogel, Ravin Balakrishnan, and Andy Cockburn. 2008. The impact of control-display gain on user performance in pointing tasks. *Human–computer interaction* 23, 3 (2008), 215–250.

[15] Mustafa Mert Çelikok, Pierre-Alexandre Murena, and Samuel Kaski. 2023. Modeling needs user modeling. *Frontiers in Artificial Intelligence* 6 (2023), 61.

[16] Yash Chandak, Georgios Theocharous, Chris Nota, and Philip Thomas. 2020. Lifelong learning with a changing action set. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3373–3380.

[17] Krishnendu Chatterjee and Laurent Doyen. 2014. Partial-observation stochastic games: How to win when belief fails. *ACM Transactions on Computational Logic (TOCL)* 15, 2 (2014), 1–44.

[18] Noshaba Cheema, Laura A Frey-Law, Kourosh Naderi, Jaakko Lehtinen, Philipp Slusallek, and Perttu Hämäläinen. 2020. Predicting mid-air interaction movements and fatigue using deep reinforcement learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[19] Xiuli Chen, Aditya Acharya, Antti Oulasvirta, and Andrew Howes. 2021. An adaptive model of gaze-based selection. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–11.

[20] Yifei Cheng, Yukang Yan, Xin Yi, Yuanchun Shi, and David Lindlbauer. 2021. SemanticAdapt: Optimization-based Adaptation of Mixed Reality Layouts Leveraging Virtual-Physical Semantic Connections. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21)*. Association for Computing Machinery, New York, NY, USA, 282–297. doi:10.1145/3472749.3474750

[21] Allan Dafoe, Edward Hughes, Yoram Bachrach, Tantum Collins, Kevin R McKee, Joel Z Leibo, Kate Larson, and Thore Graepel. 2020. Open problems in cooperative AI. *arXiv preprint arXiv:2012.08630* (2020).

[22] Alan John Dix. 1991. *Formal methods for interactive systems*. Vol. 16. Academic Press London.

[23] Anca D Dragan and Siddhartha S Srinivasa. 2013. A policy-blending formalism for shared control. *The International Journal of Robotics Research* 32, 7 (2013), 790–805.

[24] Yuqing Du, Stas Tiomkin, Emre Kiciman, Daniel Polani, Pieter Abbeel, and Anca Dragan. 2020. Ave: Assistance via empowerment. *Advances in Neural Information Processing Systems* 33 (2020), 4560–4571.

[25] Ayoub Echchahed and Pablo Samuel Castro. 2025. A Survey of State Representation Learning for Deep Reinforcement Learning. *arXiv preprint arXiv:2506.17518* (2025).

[26] Maxim Egorov, Zachary N Sunberg, Edward Balaban, Tim A Wheeler, Jayesh K Gupta, and Mykel J Kochenderfer. 2017. POMDPs. jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research* 18, 26 (2017), 1–5.

[27] Umer Farooq and Jonathan Grudin. 2016. Human-computer integration. *interactions* 23, 6 (2016), 26–32.

[28] Robert S Fenchel. 1981. An integral approach to user assistance. In *Proceedings of the Joint Conference on Easier and More Productive Use of Computer Systems.(Part-II): Human Interface and the User Interface-Volume 1981*. 98–104.

[29] Alan Fern, Sriraam Natarajan, Kshitij Judah, and Prasad Tadepalli. 2014. A decision-theoretic model of assistance. *Journal of Artificial Intelligence Research* 50 (2014), 71–104.

[30] Gerhard Fischer. 1989. Human-computer interaction software: lessons learned, challenges ahead. *IEEE Software* 6, 1 (1989), 44–52.

[31] Gerhard Fischer. 2001. User modeling in human–computer interaction. *User modeling and user-adapted interaction* 11, 1 (2001), 65–86.

[32] Paul M Fitts, MS Viteles, NL Barr, DR Brimhall, Glen Finch, Eric Gardner, WF Grether, WE Kellum, and SS Stevens. 1951. *Human engineering for an effective air-navigation and traffic-control system, and appendixes 1 thru 3*. Technical Report. Ohio State Univ Research Foundation Columbus.

[33] Michael Fleming and Robin Cohen. 2001. A user modeling approach to determining system initiative in mixed-initiative ai systems. In *International Conference on User Modeling*. Springer, 54–63.

[34] Krzysztof Gajos and Daniel S Weld. 2004. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*. 93–100.

[35] Krzysztof Z Gajos, Daniel S Weld, and Jacob O Wobbrock. 2010. Automatically generating personalized user interfaces with Supple. *Artificial Intelligence* 174, 12-13 (2010), 910–950.

[36] Samuel J Gershman, Eric J Horvitz, and Joshua B Tenenbaum. 2015. Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science* 349, 6245 (2015), 273–278.

[37] MK Ghosh, D McDonald, and S Sinha. 2004. Zero-sum stochastic games with partial information. *Journal of optimization theory and applications* 121, 1 (2004), 99–118.

[38] Blandine Ginon. 2012. Towards a generic model for user assistance. In *International Conference on User Modeling, Adaptation, and Personalization*. Springer, 356–360.

[39] Blandine Ginon. 2014. *Modèles et outils génériques pour mettre en place des systèmes d'assistance épiphytes*. Ph. D. Dissertation. INSA de Lyon.

[40] Yves Guiard, Renaud Blanch, and Michel Beaudouin-Lafon. 2004. Object pointing: a complement to bitmap pointing in GUIs. In *Proceedings of Graphics Interface 2004*. Citeseer, 9–16.

[41] Tanay Gupta and Julien Gori. 2023. Modeling reciprocal adaptation in HCI: a Multi-Agent Reinforcement Learning Approach. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–6.

[42] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. 2016. Cooperative inverse reinforcement learning. *Advances in neural information processing systems* 29 (2016).

[43] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. 2020. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193* (2020).

[44] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. 2004. Dynamic programming for partially observable stochastic games. In *AAAI*, Vol. 4. 709–715.

[45] Steve Harrison, Deborah Tatar, and Phoebe Sengers. 2007. The three paradigms of HCI. In *Alt. Chi. Session at the SIGCHI Conference on human factors in computing systems San Jose, California, USA*. 1–18.

[46] John H Holland. 1992. Complex adaptive systems. *Daedalus* 121, 1 (1992), 17–30.

[47] Kristina Höök. 2000. Steps to take before intelligent user interfaces become real. *Interacting with computers* 12, 4 (2000), 409–426.

[48] Karel Horák and Branislav Bošanskỳ. 2019. Solving partially observable stochastic games with public observations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 2029–2036.

[49] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 159–166.

[50] Eric J Horvitz. 2007. Reflections on challenges and promises of mixed-initiative interaction. *AI Magazine* 28, 2 (2007), 3–3.

[51] Ronald A Howard. 1960. Dynamic programming and markov processes. (1960).

[52] Andrew Howes, Richard L Lewis, and Alonso Vera. 2009. Rational adaptation under task and processing constraints: implications for testing theories of cognition and action. *Psychological review* 116, 4 (2009), 717.

[53] Andrew Howes, Paul A Warren, George Farmer, Wael El-Deredy, and Richard L Lewis. 2016. Why contextual preference reversals maximize expected value. *Psychological review* 123, 4 (2016), 368.

[54] PR Innocent. 1982. Towards self-adaptive interface systems. *International Journal of Man-Machine Studies* 16, 3 (1982), 287–299.

[55] Anthony Jameson. 2007. Adaptive interfaces and agents. In *The human-computer interaction handbook*. CRC press, 459–484.

[56] Alex Jansen, Leah Findlater, and Jacob O Wobbrock. 2011. From the lab to the world: Lessons from extending a pointing technique for real-world use. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. 1867–1872.

[57] Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. 2015. Shared autonomy via hindsight optimization. *Robotics science and systems: online proceedings* 2015 (2015).

[58] Frédéric Jean and Sofya Maslovskaya. 2018. Inverse optimal control problem: the linear-quadratic case. In *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 888–893.

[59] Jussi Jokinen, Aditya Acharya, Mohammad Uzair, Xinhui Jiang, and Antti Oulasvirta. 2021. Touchscreen typing as optimal supervisory control. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[60] Mayank Kabra, Alice A. Robie, Marta Rivera-Alba, Steven Branson, and Kristin Branson. 2013. JAABA: interactive machine learning for automatic annotation of animal behavior. *Nature Methods* 10, 1 (Jan. 2013), 64–67. doi:10.1038/nmeth.2281 Publisher: Nature Publishing Group.

[61] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101, 1-2 (1998), 99–134.

[62] Daniel Kahneman and Amos Tversky. 2013. Prospect theory: An analysis of decision under risk. In *Handbook of the fundamentals of financial decision*

making: Part I. World Scientific, 99–127.
[63] Lodewijk Kallenberg. 2011. Markov decision processes. *Lecture Notes. University of Leiden* 428 (2011).
[64] Tobias Kaupp, Alexei Makarenko, and Hugh Durrant-Whyte. 2010. Human–robot communication for collaborative decision making—A probabilistic approach. *Robotics and Autonomous Systems* 58, 5 (2010), 444–456.
[65] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisỳ. 2019. Rethinking formal models of partially observable multiagent decision making. *arXiv preprint arXiv:1906.11110* (2019).
[66] Christian Kroer and Tuomas Sandholm. 2015. Discretization of Continuous Action Spaces in Extensive-Form Games.. In *AAMAS*. 47–56.
[67] Akshat Kumar and Shlomo Zilberstein. 2009. Dynamic programming approximations for partially observable stochastic games. In *Twenty-Second International FLAIRS Conference*.
[68] Thomas Langerak, Sammy Christen, Mert Albaba, Christoph Gebhardt, Christian Holz, and Otmar Hilliges. 2024. MARLUI: Multi-Agent Reinforcement Learning for Adaptive Point-and-Click UIs. *Proceedings of the ACM on Human-Computer Interaction* 8, EICS (2024), 1–27.
[69] Byungjoo Lee, Mathieu Nancel, Sunjun Kim, and Antti Oulasvirta. 2020. AutoGain: Gain Function Adaptation with Submovement Efficiency Optimization. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–12.
[70] Richard L Lewis, Andrew Howes, and Satinder Singh. 2014. Computational rationality: Linking mechanism and behavior through bounded utility maximization. *Topics in cognitive science* 6, 2 (2014), 279–311.
[71] Joseph CR Licklider. 1960. Man-computer symbiosis. *IRE transactions on human factors in electronics* 1 (1960), 4–11.
[72] Henry Lieberman. 2009. User interface goals, AI opportunities. *AI Magazine* 30, 4 (2009), 16–16.
[73] Wanyu Liu, Rafael Lucas d'Oliveira, Michel Beaudouin-Lafon, and Olivier Rioul. 2017. Bignav: Bayesian information gain for guiding multiscale navigation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 5869–5880.
[74] Wanyu Liu, Olivier Rioul, Joanna Mcgrenere, Wendy E Mackay, and Michel Beaudouin-Lafon. 2018. BIGFile: Bayesian information gain for fast file retrieval. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.
[75] Yong Liu, Jorge Goncalves, Denzil Ferreira, Bei Xiao, Simo Hosio, and Vassilis Kostakos. 2014. CHI 1994-2013: Mapping two decades of intellectual progress through co-word analysis. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 3553–3562.
[76] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931* (2023).
[77] Pattie Maes. 1995. Agents that reduce work and information overload. In *Readings in human–computer interaction*. Elsevier, 811–821.
[78] Todd L McKerchar and C Renee Renda. 2012. Delay and probability discounting in humans: An overview. *The Psychological Record* 62, 4 (2012), 817–834.
[79] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.
[80] Roderick Murray-Smith, Antti Oulasvirta, Andrew Howes, Jörg Müller, Aleksi Ikkala, Miroslav Bachinski, Arthur Fleig, Florian Fischer, and Markus Klar. 2022. What simulation can do for HCI research. *Interactions* 29, 6 (2022), 48–53.
[81] Karen Myers, Pauline Berry, Jim Blythe, Ken Conley, Melinda Gervasio, Deborah L McGuinness, David Morley, Avi Pfeffer, Martha Pollack, and Milind Tambe. 2007. An intelligent personal assistant for task and time management. *AI Magazine* 28, 2 (2007), 47–47.
[82] John F Nash Jr. 1950. Equilibrium points in n-person games. *Proceedings of the national academy of sciences* 36, 1 (1950), 48–49.
[83] Andrew Y Ng, Stuart Russell, et al. 2000. Algorithms for inverse reinforcement learning.. In *Icml*, Vol. 1. 2.
[84] Jakob Nielsen. 1994. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 152–158.
[85] Stefanos Nikolaidis, Anton Kuznetsov, David Hsu, and Siddhartha Srinivasa. 2016. Formalizing human-robot mutual adaptation: A bounded memory model. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 75–82.
[86] Aurélien Nioche, Pierre-Alexandre Murena, Carlos de la Torre-Ortiz, and Antti Oulasvirta. 2021. Improving Artificial Teachers by Considering How People Learn and Forget. In *26th International Conference on Intelligent User Interfaces*. 445–453.
[87] Donald A Norman. 1986. Cognitive engineering. In *User centered system design*. CRC Press, 31–62.
[88] Donald A. Norman. 2002. *The Design of Everyday Things*. Basic Books, Inc., USA.
[89] Frans A Oliehoek and Christopher Amato. 2015. A concise introduction to decentralized POMDPs.
[90] Afshin OroojlooyJadid and Davood Hajinezhad. 2019. A review of cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1908.03963* (2019).
[91] Antti Oulasvirta, Niraj Ramesh Dayama, Morteza Shiripour, Maximilian John, and Andreas Karrenbauer. 2020. Combinatorial optimization of graphical user interface designs. *Proc. IEEE* 108, 3 (2020), 434–464.
[92] Antti Oulasvirta, Per Ola Kristensson, Xiaojun Bi, and Andrew Howes. 2018. *Computational interaction*. Oxford University Press.
[93] Alexander Pan, Kush Bhatia, and Jacob Steinhardt. 2022. The effects of reward misspecification: Mapping and mitigating misaligned models. *arXiv preprint arXiv:2201.03544* (2022).
[94] Liviu Panait and Sean Luke. 2005. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems* 11, 3 (2005), 387–434.
[95] Simon Parsons and Michael Wooldridge. 2002. Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 5, 3 (2002), 243–254.
[96] Jens G Pohl. 1997. Human-Computer Partnership in Decision-Support Systems: Some Design Guidelines. In *Proceedings of InterSymp-1997: The 9th International Conference on Systems Research, Informatics and Cybernetics: Baden-Baden, Germany*. 71.
[97] Angel R Puerta, Henrik Eriksson, John H Gennari, and Mark A Musen. 1994. Beyond data models for automated user interface generation. In *BCS HCI*. Citeseer, 353–366.
[98] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.
[99] Siddharth Reddy, Igor Labutov, Siddhartha Banerjee, and Thorsten Joachims. 2016. Unbounded human learning: Optimal scheduling for spaced repetition. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1815–1824.
[100] Yvonne Rogers. 2004. New theoretical approaches for human-computer interaction. *Annual review of information science and technology* 38, 1 (2004), 87–143.
[101] Silvia Schiaffino and Analıa Amandi. 2004. User–interface agent interaction: personalization issues. *International Journal of Human-Computer Studies* 60, 1 (2004), 129–148.
[102] Rohin Shah, Pedro Freire, Neel Alex, Rachel Freedman, Dmitrii Krasheninnikov, Lawrence Chan, Michael D Dennis, Pieter Abbeel, Anca Dragan, and Stuart Russell. 2020. Benefits of assistance over reward learning. (2020).
[103] Ben Shneiderman and Pattie Maes. 1997. Direct manipulation vs. interface agents. *interactions* 4, 6 (1997), 42–61.
[104] Yoav Shoham and Kevin Leyton-Brown. 2008. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.
[105] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. 2021. Reward is enough. *Artificial Intelligence* (2021), 103535.
[106] Aleksandrs Slivkins et al. 2019. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning* 12, 1-2 (2019), 1–286.
[107] Eilon Solan and Nicolas Vieille. 2015. Stochastic games. *Proceedings of the National Academy of Sciences* 112, 45 (2015), 13743–13746.
[108] Constantine Stephanidis, Gavriel Salvendy, Margherita Antona, Jessie YC Chen, Jianming Dong, Vincent G Duffy, Xiaowen Fang, Cali Fidopiastis, Gino Fragomeni, Limin Paul Fu, et al. 2019. Seven HCI grand challenges. *International Journal of Human–Computer Interaction* 35, 14 (2019), 1229–1269.
[109] Constantine Stephanidis, Gavriel Salvendy, Margherita Antona, Vincent G Duffy, Qin Gao, Waldemar Karwowski, Shin'ichi Konomi, Fiona Nah, Stavroula Ntoa, Pei-Luen Patrick Rau, et al. 2025. Seven HCI grand challenges revisited: Five-year progress. *International Journal of Human–Computer Interaction* (2025), 1–49.
[110] Richard S Sutton, Andrew G Barto, et al. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.
[111] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. 2021. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 15032–15043.
[112] Justin K Terry, Nathaniel Grammel, Benjamin Black, Ananth Hari, Caroline Horsch, and Luis Santos. 2020. Agent Environment Cycle Games. *arXiv preprint arXiv:2009.13051* (2020).
[113] Loren G Terveen. 1995. Overview of human-computer collaboration. *Knowledge-Based Systems* 8, 2-3 (1995), 67–81.
[114] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. 2021. Adapting user interfaces with model-based reinforcement learning. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–13.
[115] Sarah Theres Völkel, Christina Schneegass, Malin Eiband, and Daniel Buschek. 2020. What is" intelligent" in intelligent user interfaces? a meta-analysis of 25 years of IUI. In *Proceedings of the 25th international conference on intelligent user*

interfaces. 477–487.
[116] Matthew M Walsh, Kevin A Gluck, Glenn Gunzelmann, Tiffany Jastrzembski, and Michael Krusmark. 2018. Evaluating the theoretic adequacy and applied potential of computational models of the spacing effect. *Cognitive science* 42 (2018), 644–691.
[117] Bernard Widrow and Samuel D Stearns. 1985. Adaptive signal processing.
[118] Jacob O Wobbrock, Shaun K Kane, Krzysztof Z Gajos, Susumu Harada, and Jon Froehlich. 2011. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing (TACCESS)* 3, 3 (2011), 1–27.
[119] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2021. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control* (2021), 321–384.

A The turn-by-turn interaction model as a POSG

In the turn-by-turn model of assistance, a round of interaction is played in four turns with the following events:

$$s^{(0)} \rightarrow o'_U \rightarrow s^{(1)} \rightarrow a'_U \rightarrow s^{(2)} \rightarrow o'_A \rightarrow s^{(3)} \rightarrow a'_A \rightarrow s'^{(0)} \quad (9)$$

where $s^{(k)}$ is the joint state formed by the turn number k, and the states of the task, the user, and the assistant:

$$\begin{aligned} s^{(0)} &= (0, s_T, s_U, s_A) \\ s^{(1)} &= (1, s_T, s'_U, s_A) \\ s^{(2)} &= (2, s'_T, s'_U, s_A) \\ s^{(3)} &= (3, s'_T, s'_U, s'_A) \\ s'^{(0)} &= (0, s''_T, s'_U, s'_A) \end{aligned} \quad (10)$$

and where o_U and o_A, and a_U and a_A are the observation and actions respectively produced by the user and the assistant.

To cast the sequential interaction model as a POSG, we define the following joint observations

$$\begin{aligned} o^{(1)} &= (o'_U, \text{No-Op}) \\ o^{(2)} &= (\text{No-Op}, \text{No-Op}) \\ o^{(3)} &= (\text{No-Op}, o'_A) \\ o'^{(0)} &= (\text{No-Op}, \text{No-Op}) \end{aligned} \quad (11)$$

and joint actions:

$$\begin{aligned} a^{(0)} &= (\text{No-Op}, \text{No-Op}) \\ a^{(1)} &= (a'_U, \text{No-Op}) \\ a^{(2)} &= (\text{No-Op}, \text{No-Op}) \\ a^{(3)} &= (\text{No-Op}, a'_A) \end{aligned} \quad (12)$$

where No-Op is a No-Operation. The assistance model can then be modeled by a POSG, with the following transition and observation probabilities for each turn:

$$\begin{aligned} p(s^{(1)}, o^{(1)} | s^{(0)}, a^{(0)}) &= p(o^{(1)} | s^{(0)}, a^{(0)}) \cdot p(s^{(1)} | o^{(1)}, s^{(0)}, a^{(0)}) \\ &= \underbrace{p(o'_U | s^{(0)})}_{\text{user observation function}} \quad \underbrace{p(s^{(1)} | o'_U, s^{(0)})}_{\text{user internal transition function}} \end{aligned} \quad (13)$$

$$\begin{aligned} p(s^{(2)}, o^{(2)} | s^{(1)}, a^{(1)}) &= p(s^{(2)} | s^{(1)}, a^{(1)}) \\ &= \underbrace{p(s^{(2)} | s^{(1)}, a'_U)}_{\text{user external transition function}} \end{aligned} \quad (14)$$

$$\begin{aligned} p(s^{(3)}, o^{(3)} | s^{(2)}, a^{(2)}) &= p(o^{(3)} | s^{(2)}, a^{(2)}) \cdot p(s^{(3)} | o^{(3)}, s^{(2)}, a^{(2)}) \\ &= \underbrace{p(o'_A | s^{(2)})}_{\text{assistant observation function}} \quad \underbrace{p(s^{(3)} | o'_A, s^{(2)})}_{\text{assistant internal transition function}} \end{aligned} \quad (15)$$

$$\begin{aligned} p(s'^{(0)}, o'^{(0)} | s^{(3)}, a^{(3)}) &= p(s'^{(0)} | s^{(3)}, a^{(3)}) \\ &= \underbrace{p(s'^{(0)} | s^{(3)}, a'_A)}_{\text{assistant external transition function}} \end{aligned} \quad (16)$$

Rewards are simply collected at each function and distributed to the relevant agent. The use of No-Ops is needed to map the turn-based interaction model to the simultaneous nature of POSGs. This argument can be made more rigorous [111, 112]. Similarly, No-Ops can be used to model more complex interaction modes, *e.g.* by having agents skip rounds, also see [111, 112].

The sequential interaction model can thus be turned to a POSG, where the transition and observation function $P(s', o|s, a)$ admits a different expression for each turn. The transition and observation function can be reduced into elementary functions of observation and inference (internal transition) (Eqs. 13 and 15), and task state transition (external transition) (Eqs. 14 and 16). These are the elementary functions that define the common basis to model interactive contexts on top of which we build our library.