Task Planning of Cyber-Human Systems

Roykrong Sukkerd[1(✉)], David Garlan[1], and Reid Simmons[2]

[1] Institute for Software Research, School of Computer Science Carnegie Mellon University, Pittsburgh, PA, USA
{rsukkerd,garlan}@cs.cmu.edu

[2] Robotics Institute, School of Computer Science Carnegie Mellon University, Pittsburgh, PA, USA
reids@cs.cmu.edu

Abstract. Internet of Things (IoT) allows for cyber-physical applications to be created and composed to provide intelligent support or automation of end-user tasks. For many of such tasks, human participation is crucial to the success and the quality of the tasks. The cyber systems should proactively request help from the humans to accomplish the tasks when needed. However, the outcome of such system-human synergy may be affected by factors external to the systems. Failure to consider those factors when involving human participants in the tasks may result in suboptimal performance and negative experience on the humans. In this paper, we propose an approach for automated generation of control strategies of cyber-human systems. We investigate how explicit modeling of human participant can be used in automated planning to generate cooperative strategy of human and system to achieve a given task, by means of which best and appropriately utilize the human. Specifically, our approach consists of: (1) a formal framework for modeling cooperation between cyber system and human, and (2) a formalization of system-human cooperative task planning as strategy synthesis of stochastic multiplayer game. We illustrate our approach through an example of indoor air quality control in smart homes.

Keywords: Cyber-human systems · Planning · Stochastic multiplayer games

1 Introduction

Computing has become increasingly ubiquitous and integrated into our daily lives through interconnected devices and services inhabiting in our living environments. The advancement of this ubiquitous computing paradigm enables us to automate processes to support our everyday living and activities, such as efficient home heating/cooling, security and emergency response, and navigation. We refer to these processes as *tasks*.

This work is supported by Bosch Research and Technology Center North America.

R. Calinescu and B. Rumpe (Eds.): SEFM 2015, LNCS 9276, pp. 293–309, 2015.
DOI: 10.1007/978-3-319-22969-0_21

Cyber and robotic systems that can carry out our everyday tasks in a fully autonomous way are still far from reality. In many cases, human involvement is crucial to the success and the quality of tasks. Some tasks may require manual steps – humans may need to act, provide information, or make decisions for the cyber or robotic systems responsible for those tasks. For example, an emergency response system requires human responders to provide first aid to the patients. And even when human involvement is not strictly required, many tasks can be performed better with cooperation from humans. For example, a navigation system may obtain information about occurring events around the area from the locals to plan a better tour for the visitors.

It is important, therefore, to consider *cyber-human system* (CHS) paradigm, in which cyber systems in smart environments cooperate with humans to carry out tasks. Such cyber-human systems raise a number of challenges for software and system engineers, who must decide when and how humans should be involved, how to deal with the uncertainty inherent in having humans in the loop, how to provide assurances that such systems will not go awry, and how to take into consideration variability of human capability and motivation for task participation. Moreover, ideally engineered solutions should be flexible enough to accommodate at low cost the rapidly changing contexts of smart environments in which mobile users move in and out of spaces, new tasks are introduced, and new devices, technologies, appliances and services may become available at any time.

Unfortunately, today's practice fails to adequately address these challenges. Software for smart environments tends to be written as handcrafted programs for particular environments by specialized engineers, making it costly to create and even more costly to upgrade. Code tends to involve complex low-level logic encoded as if-then-else statements, which is brittle, hard to maintain, and difficult to validate. Policies for coordination between automated and non-automated functions tend to be wired in, and are largely context independent. In cases where systems adapt to context (such as the Nest smart thermostat), they tend to be isolated devices or subsystems, and have limited ability to explicitly leverage human capabilities.

What is missing is a way to describe tasks and develop strategies for accomplishing them that (a) provides flexibility by accommodating varying contexts, human factors, and changing technology, (b) accounts for the inherent uncertainty in human-in-the-loop systems, (c) can be analyzed formally to provide probabilistic guarantees about the expected quality of a plan under various conditions.

To this end, we explore the use of automated planning to design cooperation between cyber systems and humans in performing tasks, in which human participants have the role of actuators. We investigate how explicit modeling of human participants can be used in planning to generate cooperative strategies which best and appropriately utilize the humans. Specifically, our contributions are: (1) a formal framework for modeling cooperation between cyber systems and humans under uncertainty, with an explicit modeling of human participants

based on opportunity-willingness-capabiity (OWC) ontology, and (2) a formalization of cyber system-human cooperative task planning as strategy synthesis of stochastic multiplayer game.

This paper is organized as follows. Section 2 provides background on SMG. Section 3 describes the running example. Section 4 presents the approach to model system-human delegation. Section 5 presents the formalization of system-human cooperative task planning as strategy synthesis of SMG. Section 6 shows the analysis results from the running example. Section 7 discusses the related work and Sect. 8 concludes the paper.

2 Preliminaries

This section introduces our notion of tasks, and background on stochastic multiplayer games (SMGs) and strategy synthesis of SMGs – the technique on which we build our approach.

2.1 Task Representation

We consider a task to consist of a *reachability goal* and a *utility function*, denoted as $T = \langle goal, r \rangle$. *goal* is a predicate describing the end condition of T. This end condition may be the desired condition that T must achieve, or it may simply indicate the end of T's duration. $r : S \to \mathbb{R}_{\geq 0}$ is the utility function that maps the states of the execution context to their associated utility values. This utility function captures the qualities of concern of T and allows for trade-offs among multiple potentially conflicting objectives over those qualities. The utility function is to be optimized in the task planning.

2.2 Stochastic Multiplayer Games

A turn-based stochastic multiplayer game (SMG) is a tuple $\mathcal{G} = \langle \Pi, S, A, (S_i)_{i \in \Pi}, \Delta, AP, \chi \rangle$, where: Π is a finite set of players; S is a finite, non-empty set of states; A is a finite, non-empty set of actions; $(S_i)_{i \in \Pi}$ is a partition of S; $\Delta : S \times A \to \mathcal{D}(S)$ is a partial transition function; AP is a finite set of atomic propositions; and $\chi : S \to 2^{AP}$ is a labeling function.

In each state $s \in S$ of the SMG $\mathcal{G}$, the set of available actions is denoted by $A(s) := \{a \in A | \Delta(s, a) \neq \perp\}$. We assume that $A(s) \neq \emptyset$ for all s. The choice of action to take in s is under the control of exactly one player, namely the player $i \in \Pi$ for which $s \in S_i$. Once action $a \in A(s)$ is selected, the successor state is chosen according to the probability distribution $\Delta(s, a)$.

A path of SMG $\mathcal{G}$ is an (in)finite sequence $\lambda = s_0 a_0 s_1 a_1 ...$ such that $\forall j \in \mathbb{N} \cdot a_j \in A(s_j) \wedge \Delta(s_j, a_j)(s_{j+1}) > 0$. $\Omega_{\mathcal{G}}^{+}$ denotes the set of finite paths in $\mathcal{G}$.

A strategy for player $i \in \Pi$ in $\mathcal{G}$ is a function $\sigma_i : (SA)^* S_i \to \mathcal{D}(A)$ which, for each path $\lambda \cdot s \in \Omega_{\mathcal{G}}^{+}$ where $s \in S_i$, selects a probability distribution $\sigma_i(\lambda \cdot s)$ over $A(s)$.

2.3 Strategy Synthesis of SMGs

Reasoning about strategies is a fundamental aspect of model checking SMGs, which enables checking for the existence of a strategy that is able to optimize an objective expressed as a quantitative property in a logic called rPATL, which extends ATL, a logic extensively used to reason about the ability of a set of players to collectively achieve a particular goal. Properties written in rPATL can state that a coalition of players has a strategy which can ensure that the probability of an event's occurrence or an expected reward measure meet some threshold. rPATL is a CTL-style branching-time temporal logic that incorporates the coalition operator $\langle\langle C\rangle\rangle$ of ATL, combining it with the probabilistic operator $P_{\bowtie q}$ and path formulae from PCTL. Moreover, rPATL includes a generalization of the reward operator $R^r_{\bowtie x}$ ([1,2]) to reason about goals related to rewards. An example of typical usage combining coalition and reward operators is $\langle\langle\{1,2\}\rangle\rangle R^r_{\geq 5}[F\phi]$ meaning that "players 1 and 2 have a strategy to ensure that the reward r accrued along paths leading to states satisfying state formula ϕ is at least 5, regardless of the strategies of other players." Moreover, extended versions of the rPATL reward operator $\langle\langle C\rangle\rangle R^r_{max=?}[F\phi]$ and $\langle\langle C\rangle\rangle R^r_{min=?}[F\phi]$, enable the quantification of the maximum and minimum accrued reward r along paths that lead to states satisfying ϕ that can be guaranteed by players in coalition C, independently of the strategies followed by the rest of players. Model checking of rPATL properties supports optimal strategy synthesis for a given property.

3 Running Example

Air quality control system (AQC-sys) periodically monitors the air quality in the home – a measure of how clean or polluted the air is, as indicated by the index ranging *bad*, *moderate*, and *good* – and controls it to be at a desirable level. AQC-sys can clean the air by running an electric air purifier. Alternatively, if the condition of the outdoor climate is favorable (e.g., no pollution and the temperature is desirable), the indoor air quality can be improved by means of wind ventilation through open windows. However, AQC-sys does not have a mechanism to directly control the windows in the home; therefore, it has to request the occupant to open the windows for ventilation.

There are 3 concerns regarding this task:

1. Air quality – When the home is occupied, higher air quality is always preferred to lower air quality. When the home is vacant, the air qualities *moderate* and *good* are equally desirable, and are preferred to the air quality *bad*.
2. Energy consumption – Running air purifier consumes energy, while wind ventilation does not. AQC-sys should be energy-efficient.
3. Human annoyance – The occupant may get annoyed if AQC-sys requests her to open the windows when she is not willing to do so (e.g., the occupant is busy with other activity). AQC-sys should avoid being intrusive to the occupant.

Additionally, there is uncertainty due to the occupant's involvement in the task. When the occupant will be at home is uncertain, which affects when AQC-sys can request for the windows to be opened for ventilation. The occupant's willingness to cooperate with AQC-sys is also uncertain – the occupant may agree or refuse to open the windows when AQC-sys requests her to, and factors such as whether she is busy with other activity may affect that outcome.

The strategy of AQC-sys to control the air quality must make trade-off among the 3 concerns outlined above and must consider the uncertainty due to the human involvement. We will use this example throughout the paper to illustrate our approach.

4 System-Human Delegation Model

Cooperation between the cyber system and the human occurs through *delegation*. The system makes decisions about how to perform a given task, and it may request help from the human to perform some sub-tasks along the way. This section describes our approach to model system-human delegation.

4.1 Delegation

Delegation is an action denoted as $Delegate(A, B, \tau)$, where A is the delegator, B is the delegatee, and τ is the task being delegated. For simplicity, we only consider τ to be a goal with no utility function. Thus, the performance of τ refers to whether or not the goal is achieved.

To specify $Delegate(A, B, \tau)$, we need to define the state space of the delegation context, and the precondition and effect of the delegation.

Delegation Context. The precondition and effect of delegation are defined in terms of the states of the *delegation context*. The delegation context is an abstraction of the state of A, B, and their environment, at time of delegation. We denote the state space of the delegation context as S_{del}.

Precondition. The precondition of $Delegate(A, B, \tau)$, denoted as pre, is a necessary condition under which A may delegate τ to B. That is, action $Delegate(A, B, \tau)$ is applicable in $s \in S_{del}$ if and only if $s \models pre$.

Effect. The effect of $Delegate(A, B, \tau)$ is modeled as a *performance function* $f_{perf} : \bar{S}_{del} \rightarrow B$, where $\bar{S}_{del} \subseteq S_{del}$ is the set of all states $s \models pre$, and B is a set of Bernoulli random variables, each representing the (binary) outcome of τ's performance. The performance function f_{perf} represents A's *belief* about the outcomes of delegation, i.e., B's performance of τ, in different states of the delegation context. Since the performance of τ is binary, its outcome can be modeled as a Bernoulli random variable, whose success probability is the probability that A believes B will perform τ as a result of delegation.

The system may request the human to cooperate in performing the task by means of delegation. In addition to its direct actions (i.e., the actions that

affect the environment directly), the system has a set of delegation actions of the form $Delegate(sys, hum, \tau)$, where τ is a sub-task that the system may need the human to perform. Identifying the precondition and effect of delegation action requires knowing what factors affect the human's performance of the task and in what way.

Next, we discuss the approach to model the precondition and effect of system-human delegation, and illustrate it through the running example.

4.2 Human Model

To capture factors that may influence the human's performance of a task and how, we employ the opportunity-willingness-capability (OWC) ontology [5], which classifies a set of factors on which the task's performance is conditioned. For each task τ which the system may want to delegate to the human, we create an OWC model by identifying the factors for each of the following categories:

Opportunity. Opportunity captures the prerequisites for task performance. *Opportunity elements* (OE) are variables relevant to such prerequisites. *Opportunity function* f_O is a boolean formula of the states of OE, determining whether the task performance is possible.

Running Example: The opportunity of window-opening task is that the occupant must be at home. The opportunity element is OE = $\{occupant\_at\_home\}$. The opportunity function is $f_O = (occupant\_at\_home == true)$.

Willingness. Willingness captures the desire of the human to perform the task. *Willingness elements* (WE) are variables that influence such desire. *Willingness function* is $f_W : S_{WE} \rightarrow B$, where S_{WE} is the state space of WE and B is a set of Bernoulli random variables. The success probability of each Bernoulli random variable $x \in B$ represents the *willingness probability* p_W associated with the state of WE to which x is mapped.

Running Example: The willingness of window-opening task is influenced by whether the occupant is busy. The willingness element is WE = $\{occupant\_busy\}$. The willingness probabilities for when *occupant_busy* and $\neg occupant\_busy$ are 0.1 and 0.95, respectively.

Capability Capability captures the humans' ability to perform the task, given opportunity and willingness. *Capability elements* (CE) are variables that influence such ability. Similar to willingness, *capability function* is $f_C : S_{CE} \rightarrow B$, where S_{CE} is the state space of CE and B is a set of Bernoulli random variables. The success probability of each Bernoulli random variable $x \in B$ represents the *capability probability* p_C associated with the state of CE to which x is mapped.

Running Example: For window-opening task, we assume that the capability is trivial, i.e., $CE = \emptyset$.

4.3 From OWC Model to Delegation Model

We use the OWC model of task τ to derive the specification of $Delegate(sys, hum, \tau)$ as follow:

Delegation Context. We represent the delegation context of $Delegate(sys, hum, \tau)$ using the OWC elements, i.e., defining S_{del} to be the state space of $OE \cup WE \cup CE$.

Precondition. We define the precondition pre of $Delegate(sys, hum, \tau)$ to be the opportunity function f_O of the OWC model of τ.

Effect. The effect of $Delegate(sys, hum, \tau)$, i.e., the performance function f_{perf}, is derived from the OWC model of τ as follow. Recall that $f_{perf} : \bar{S}_{del} \rightarrow B$. For each state $s \in \bar{S}_{del}$, $f_{perf}(s)$ is a Bernoulli random variable with success probability $p_W \cdot p_C$, where p_W and p_C are the willingness and the capability probabilities associated with WE and CE components of s, respectively.

Running Example: For window-opening task τ, S_{del} is the state space built over the state variables *occupant_at_home* and *occupant_busy*. The precondition of $Delegate(sys, hum, \tau)$ is: $occupant\_at\_home == true$. The effect of $Delegate(sys, hum, \tau)$ is: if $occupant\_at\_home \wedge occupant\_busy$, the probability of τ's performance is 0.1; if $occupant\_at\_home \wedge \neg occupant\_busy$, the probability of τ's performance is 0.95. Otherwise, the effect of $Delegate(sys, hum, \tau)$ is undefined.

5 System-Human Cooperative Task Planning

In this section, we present a formalization of system-human cooperative task planning problem as strategy synthesis of SMG. We also provide a description of our running example's SMG model implemented in the probabilistic model-checker PRISM-games [4].[1]

5.1 SMG Model

The SMG representing the interactions among the cyber system, the human, and the environment is $\mathcal{G} = \langle \Pi, S, A, (S_i)_{i \in \Pi}, \Delta, AP, \chi, r \rangle$, where:

- $\Pi = \{sys, hum, env\}$ is the set of players, representing the system, the human, and the environment.
- $S = S_{sys} \cup S_{hum} \cup S_{env}$ is the set of states, where S_{sys}, S_{hum}, and S_{env} are the states controlled by the players sys, hum, and env, respectively, and $S_{sys} \cap S_{hum} \cap S_{env} = \emptyset$.

[1] We illustrate our approach to modeling the SMG using the syntax of the PRISM language [3] for SMG, which are encoded as commands:

$$[action]\ guard -> p_1 : u_1 + ... + p_n : u_n$$

where $guard$ is a predicate over the model variables. Each update u_i describes a transition that the process can make (by executing $action$) if the guard is true. An update is specified by giving the new values of the variables, and has an assigned probability $p_i \in [0, 1]$. Multiple commands with overlapping guards (and probably, including a single update of unspecified probability) introduce local nondeterminism.

- $A = A_{sys} \cup A_{hum} \cup A_{env}$ is the set of actions, where A_{sys}, A_{hum}, and A_{env} are the actions available to the players *sys*, *hum*, and *env*, respectively.
- $r : S \to \mathbb{R}_{\geq 0}$ is the utility function capturing the qualities of concern of the task.

Players *sys*, *hum*, and *env* take alternating turns of the control of the game. We use a special state variable *turn* to distinguish between the states S_{sys}, S_{hum}, and S_{env}. When there is no delegation, the control of the game evolves in a round-robin fashion: the control is transferred from *env* to *hum*, to *sys*, and back to *env*. When there is delegation, instead of yielding the control to *env*, *sys* yields the control to *hum*. Next, if the delegated task is successfully performed, then *hum* yields the control to *env*. Otherwise, the delegated task is not performed and *hum* yields the control to *sys*. Next, *sys* yields the control to *env*, and the transfer of the control goes back to the round-robin fashion until the next delegation.

To incorporate our system-human delegation model in $\mathcal{G}$, we must first include, in the set of state variables that define S, all the OWC elements associated with all the tasks which the system may delegate to the human.

Running Example: There is only 1 task which AQC-sys may delegate to the occupant: opening the windows. The OWC elements for the task are *occupant_at_home* and *occupant_busy*.

5.2 Environment

Player *env* controls the actions A_{env} available in S_{env}. Conceptually, *env* models potential occurrences of events that are out of the system's and the human's control. Each action $a \in A_{env}$ available in a state $s \in S_{env}$ updates 0 or more environment variables and always yields the control of the game to player *hum*.

Running Example: Player *env* models the evolution of time, and the effects of running the air purifier and wind ventilation on the indoor air quality. The game ends when the time reaches the defined planning horizon. We simplify the running example by assuming that within the planning horizon, the outdoor air quality remains constant and the indoor air quality does not decrease.[2]

```
1  module environment
2    t : [0..MAX_TIME] init 0;
3    aqi_out : [GOOD..BAD] init GOOD;
4    aqi_in : [GOOD..BAD] init MODERATE;
5
6    // effect of running air purifier
7    [purify] turn=ENV & t<MAX_TIME & purifier_on -> (aqi_in'=GOOD) & (t'=t+TAU) & (turn'=
         HUM);
8
9    // effect of wind ventilation
10   [vent] turn=ENV & t<MAX_TIME & window_open -> (aqi_in'=aqi_out) & (t'=t+TAU) & (turn
         '=HUM);
11
```

[2] In this example, player *env* only has deterministic behavior. However, in general, it can have probabilistic and nondeterministic behavior as well.

```
12   // no change in air quality
13   [env_none] turn=ENV & t<MAX_TIME & !purifier_on & !window_open -> (t'=t+TAU) & (turn
         '=HUM);
14 endmodule
```

Listing 1.1. Environment module

Listing 1.1 shows the encoding of the environment. The variable `t` (line 2) keeps track of time, which increments by a discrete value `TAU` (e.g., 10 min). The variables `aqi_out` and `aqi_in` (line 3 and line 4) represent the indices of the outdoor and indoor air quality, respectively. The transition `purify` (line 7) models the effect of running the air purifier on the indoor air quality – if the air purifier is turned on, then in the next `TAU`, the indoor air quality will be at level `GOOD`. Similarly, the transition `vent` (line 10) models the effect of wind ventilation on the indoor air quality – if the windows are open, then in the next `TAU`, the indoor air quality will be at the same level as that of the outdoor air quality. The transition `env_none` (line 13) models the indoor air quality when the air purifier is off and the windows are closed – no change. All transitions yield the turn to player *hum* (line 7, 10, and 13).

5.3 System

Player *sys* controls the actions of the system A_{sys} available in S_{sys}. A_{sys} consists of 2 disjoint subsets: direct actions A_{sys_dir}, and delegation actions A_{sys_del}.

To represent delegation, we use a special state variable *delegation*. Let the system have k tasks $\tau_1, \tau_2, ..., \tau_k$ that it may delegate to the human, we have that:

1. $delegation \in \{\varnothing, \tau_1, \tau_2, ..., \tau_k\}$, where $delegation = \varnothing$ means that the system is not currently delegating any task, and $delegation = \tau_i$ means that the system is delegating τ_i to the human.
2. $A_{sys\_del} = \{\hat{a}_1, \hat{a}_2, ..., \hat{a}_k\}$, where $\hat{a}_i$ sets $delegation = \tau_i$. That is, $\hat{a}_i$ represents $Delegate(sys, hum, \tau_i)$.

The precondition of each $\hat{a}_i \in A_{sys\_del}$ is the opportunity function f_O of the OWC model of τ_i. Thus, if $\hat{a}_i$ is available in $s \in S_{sys}$, then $s \models f_O^{\tau_i}$. However, we also want to avoid scenarios in which the system keeps delegating the same task to the human after they failed to perform it. One way to achieve this is to set a bound on the maximum number of times the system can delegate each task to the human. That is, for all $s \in S_{sys}$, $\hat{a}_i \in A_{sys\_del}$ is available in s if and only if $s \models f_O^{\tau_i}$ *and* the count on the number of times the system has delegated τ_i to the human is less than the bound.

If player *sys* chooses a delegation action $\hat{a} \in A_{sys\_del}$, the control of the game is yielded to player *hum*, i.e., the next state of the game is $s' \in S_{hum}$. Otherwise, if it chooses a direct action $a \in A_{sys\_dir}$, the control of the game is yielded to player *env*, i.e., the next state of the game is $s' \in S_{env}$.

Running Example: Player *sys* has 3 direct actions: turn on and turn off the air purifier, and do nothing; and a delegation action to request the occupant to open the windows.

```
1 module aqc_system
2   purifier_on : bool init false;
3   delegation : [0..OPEN_WINDOW] init 0;
4   count : [0..MAX_COUNT] init 0;
5
6   // turn on/off air purifier
7   [turn_on] turn=SYS & !purifier_on -> (purifier_on'=true) & (turn'=ENV);
8   [turn_off] turn=SYS & purifier_on -> (purifier_on'=false) & (turn'=ENV);
9
10  // delegate task OPEN_WINDOW
11  [delegate] turn=SYS & count<MAX_COUNT & occupant_at_home -> (delegation'=OPEN_WINDOW)
        & (count'=count+1) & (turn'=DEL);
12
13  // do nothing
14  [sys_none] turn=SYS -> (turn'=ENV);
15 endmodule
```

Listing 1.2. System module

Listing 1.2 shows the encoding of AQC-sys. The variable `purifier_on` (line 2) represents whether the air purifier is turned on and running. The variable `delegation` (line 3) represents the currently delegated task – either `OPEN_WINDOW` or none. The variable `count` (line 4) keeps track of the number of times AQC-sys has delegated the task `OPEN_WINDOW` to the occupants. The transitions `turn_on` (line 7) and `turn_off` (line 8) model the actions of AQC-sys to turn on and turn off the air purifier, respectively. The transition `delegate` (line 11) models the delegation of `OPEN_WINDOW`. This transition can only occur when there is an opportunity for `OPEN_WINDOW` and AQC-sys has not exceeded the maximum number of times it can delegate `OPEN_WINDOW` to the occupant. The transitions `turn_on`, `turn_off`, and `sys_none` yield the turn to player *env* (line 7, 8, and 14). The transition `delegate` yields the turn to player *hum* (line 11).

5.4 Human

Player *hum* controls the actions A_{hum} available in S_{hum}. Conceptually, *hum* models potential human actions and changes in the human's physical and cognitive states. We model the behavior of human when there is no delegation (i.e., the human behaves independently of the system) as well as when there is delegation. To this end, we partition S_{hum} into 2 disjoint subsets: S_{DEL} and S_{HUM}, representing the states in which the system is delegating a task to the human, and the states in which there is no delegation, respectively.

When player *hum* gains the control of the game from player *env*, the game is always in a state $s \in S_{HUM}$. Each state $s \in S_{HUM}$ has 1 or more available actions $a \in A_{hum}$. These actions always yield the control of the game to player *sys*.

When player *hum* gains the control of the game from player *sys* (that is, the system decided to delegate a task τ to the human), the game is in a state $s \in S_{DEL}$, where $s \models delegation = \tau$. We model the effect of delegation $Delegate(sys, hum, \tau)$ as follow:

Let the system have k tasks $\tau_1, \tau_2, ..., \tau_k$ that it may delegate to the human, we have the set of actions $\dot{A}_{hum} \subset A_{hum}$, where $\dot{A}_{hum} = \{\dot{a}_1, \dot{a}_2, ..., \dot{a}_k\}$. Each $\dot{a}_i \in \dot{A}_{hum}$ represents the human's performance of τ_i. For each $s \in S_{DEL}$ in which

$s \models \mathit{delegation} = \tau_i$, the only action available in s is $\dot{a}_i$. $(s, \dot{a}_i)$ probabilistically transitions to either:

1. $s' \in S_{env}$, where $s' \models \mathit{performed}(\tau)$. That is, the human successfully performed τ, and the control of the game is yielded to player *env*.
2. $s'' \in S_{sys}$, where $s'' \not\models \mathit{performed}(\tau)$ That is, the human failed to perform τ, and the control of the game is yielded to player *sys*.

The probabilities of $(s, \dot{a}_i)$ transitioning to s' and s'' are obtained from the performance function $f_{perf}^{\tau_i}$ in the system-human delegation model of τ_i.

Running Example: Player *hum* has probabilistic as well as nondeterministic behavior. The probabilistic behavior models the schedule of the occupant – when they are at home and when they are busy, and the effect of window-opening delegation. The nondeterministic behavior models the occupant's decision to open and close the windows (when it was not requested by AQC-sys).

```
1 module human
2   occupant_at_home : bool init false;  // opportunity element
3   occupant_busy : bool init false;  // willingness element
4   upd : bool init false;
5
6   // at t=0, occupant is more likely to be at home and busy
7   [t0] turn=HUM & t=0 & !upd ->
8      0.6 : (occupant_at_home'=true) & (occupant_busy'=true) & (upd'=true) +
9      0.2 : (occupant_at_home'=true) & (occupant_busy'=false) & (upd'=true) +
10     0.2 : (occupant_at_home'=false) & (upd'=true);
11
12  // at t=10, ...
13  [t10] turn=HUM & t=10 & !upd ->
14  ...
15
16  // open/close windows
17  [open] turn=HUM & upd & occupant_at_home & !window_open -> (window_open'=true) & (upd
        '=false) & (turn'=SYS);
18  [close] turn=HUM & upd & occupant_at_home & window_open -> (window_open'=false) & (upd
        '=false) & (turn'=SYS);
19
20  // do nothing
21  [hum_none] turn=HUM_TURN & upd -> (upd'=false) & (turn'=SYS);
22
23  // receive OPEN_WINDOW when not busy
24  [receive1] turn=DEL & delegation=OPEN_WINDOW & occupant_at_home & !occupant_busy ->
25     0.95 : (window_open'=true) & (turn'=ENV) +
26     0.05 : (turn'=SYS);
27
28  // receive OPEN_WINDOW when busy
29  [receive2] turn=DEL & delegation=OPEN_WINDOW & occupant_at_home & occupant_busy ->
30     0.2 : (window_open'=true) & (turn'=ENV) +
31     0.8 : (turn'=SYS);
32 endmodule
```

Listing 1.3. Human module

Listing 1.3 shows the encoding of the human. `turn=HUM` indicates state in S_{HUM} and `turn=DEL` indicates state in S_{DEL}. The variables `occupant_at_home` (line 2) and `occupant_busy` (line 3) represent whether the occupant is at home

(the opportunity element of `OPEN_WINDOW`) and whether the occupant is busy (the willingness element of `OPEN_WINDOW`), respectively. Finally, the variable `upd` (line 4) is a flag indicating whether `occupant_at_home` and `occupant_busy` are updated for the current time `t`.

When `turn=HUM`, player *hum* makes a move in 2 steps. First, it updates the state of `occupant_at_home` and `occupant_busy` at the current time `t`, based on some prediction. The transition `t0` (line 7 - 10) encodes the probability distribution over the possible states of `occupant_at_home` and `occupant_busy` at time `t=0` – there is 0.6 probability that the occupant is at home and busy (line 8), 0.2 probability that the occupant is at home and not busy (line 9), and 0.2 probability that the occupant is not at home (line 10). Once the transition `t0` is taken, the state of `occupant_at_home` and `occupant_busy` at time `t=0` become known – `upd` is set to true.

Second, player *hum* nondeterministically chooses among actions opening the windows (line 16), closing the windows (line 17), and do nothing (line 20). These actions yield the turn to player *sys*.

When `turn=DEL`, player *hum* makes a move in 1 step. The transition `receive1` (line 20 - 22) models the effect of `OPEN_WINDOW` delegation when the occupant is at home and not busy – there is 0.95 probability that the windows get opened. Similarly, the transition `receive2` (line 25 - 27) models the effect of `OPEN_WINDOW` delegation when the occupant is at home and busy. These transitions yield the turn to player *env* if `OPEN_WINDOW` is successful (line 24 and 29), and to player *sys* otherwise (line 25 and 30).

5.5 Utility Function

Utility function of the task $r : S \to \mathbb{R}$ assigns rewards to states of the system-human-environment. It is designed to incentivize certain kinds of behavior of the system. Utility function is sensitive to the context and allows for trade-offs among multiple potentially conflicting objectives, concerning different qualities such as performance and cost. In addition to the qualities associating with the task and the system, the utility function must capture qualities regarding the human's experience in working with the system, such as annoyance, cognitive and physical loads.

Running Example: The utility function defines the relative costs associated with the indoor air quality, the energy consumption, and the annoyance of the occupant. We define annoyance to be when AQC-sys requests the occupant to open the windows but the occupant refuses. The objective of the task is to minimize this utility function.

Table 1 shows the relative costs associated with different levels of the indoor air quality, per a time period of `TAU` (e.g., 10 min). The cost of indoor air quality is sensitive to whether or not the occupant is at home (e.g., the cost of bad air quality is higher when the occupant is at home than when he/she is not). Table 2 shows the relative costs associated with energy consumption of running the air purifier and wind ventilation per `TAU`, and annoyance of the occupant per each refusal.

Table 1. Costs associated with indoor air quality per 10 min

	Whether occupant is at home	
Air quality index	*occupant_at_home*	¬*occupant_at_home*
good	0	0
moderate	10	0
bad	30	2

Table 2. Costs associated with energy consumption and human annoyance

Running air purifier	15 (per 10 min)
Wind ventilation	0 (per 10 min)
Annoyance of occupants	2 (per each refusal)

To augment SMG model with utility function, we assign numeric utility values to the states of SMG. Listing 1.4 shows the encoding of the utility function, as defined in Tables 1 and 2. To define utility values for a duration (e.g., cost of air quality per 10 min, cost of running air purifier per 10 min), we assign those utility values to states in which player *env* controls the game, since we use player *env* to model the evolution of time.

```
rewards"total_cost"
  turn=ENV & aqi_in=GOOD & occupant_at_home : 0;
  turn=ENV & aqi_in=MODERATE & occupant_at_home : 10;
  turn=ENV & aqi_in=BAD & occupant_at_home : 30;

  turn=ENV & aqi_in=GOOD & !occupant_at_home : 0;
  turn=ENV & aqi_in=MODERATE & !occupant_at_home : 0;
  turn=ENV & aqi_in=BAD & !occupant_at_home : 2;

  turn=ENV & purifier_on : 15;
  turn=ENV & window_open : 0;

  turn=DEL & occupant_annoyed : 2;
endrewards
```

Listing 1.4. Utility function

5.6 SMG Strategy Synthesis

To generate a system-human cooperative plan for a task $T = \langle goal, r \rangle$, we use model checking of rPATL property to synthesize a strategy in $\mathcal{G}$ for player *sys* that has the objective of reaching a state satisfying *goal* and optimizes for the utility function r.[3]

[3] We do not generate strategies for a coalition of players *sys* and *hum* because, in addition to the cooperative behavior between the human and the system, we also want the planning model to capture the human behavior that is independent of the system. Such behavior can also affect how the task must be performed.

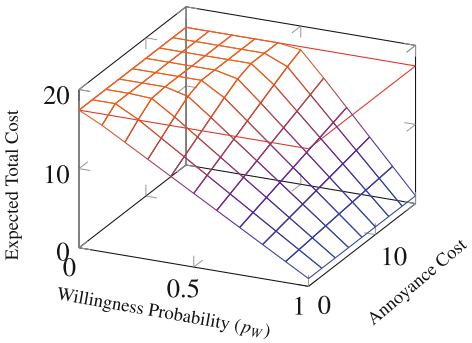

Fig. 1. Expected total costs of system-human strategies vs. system-only strategy

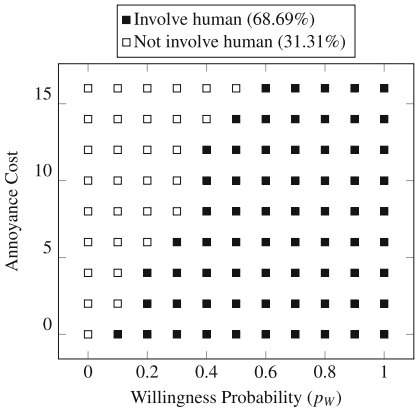

Fig. 2. Decisions of system-human strategies

The rPATL property specification for strategy synthesis is in the form $\langle\langle sys\rangle\rangle R^r_{max=?}[F\ goal]$, or alternatively $\langle\langle sys\rangle\rangle R^r_{min=?}[F\ goal]$. The resulting strategy may include delegation actions, representing the cooperation between the human and the system.

Running Example: The goal of the air quality control task is defined as a fixed time horizon `t=MAX_TIME`, and the utility function is the "total_cost" as defined in List 1.4. The rPATL property representing the task is:

$$u_{min} = \langle\langle sys\rangle\rangle\, R^{\text{“}total\_cost\text{”}}_{min=?}[F\ \texttt{t=MAX\_TIME}]$$

where u_{min} is the minimum expected utility of the generated strategy.

6 Results

In this section, we demonstrate how our SMG approach makes decisions about how to perform task in cyber-human system – especially, how decisions on delegation are made.

We implement SMG of cyber-human system for the indoor air quality control task, as described in the running example. However, instead of constant values, we vary the values of willingness probability (p_W)[4] and annoyance cost, with values in the range [0,1] and [0,16], respectively. We synthesize system-human strategies for all pairs of p_W and annoyance cost values. Additionally, we synthesize a system-only strategy, which does not have delegation action. Figure 1 shows the expected total costs of system-human and system-only strategies, for all values of p_W and annoyance cost. Figure 2 shows the decisions of system-human strategies on whether to involve human in performing the task.

[4] To simplify the analysis, we use a single value of p_W for both when the occupant is and is not busy.

In Fig. 1, the expected total cost of the system-only strategy (represented by the plane) is 17.4, independently of p_W and annoyance cost, and it is the upper bound of the expected total costs of the system-human strategies (represented by the curved surface) for all values of p_W and annoyance cost. The region of p_W-annoyance cost in which the expected total costs of system-only and system-human strategies are equal are the region in which system-human strategies do not use delegation (shown as white dots in Fig. 2) – AQC-sys always runs air purifier when it needs to improve the air quality. Whereas in the rest of the region, system-human strategies use delegation (shown as black dots in Fig. 2) when the opportunity exists.

Decision on whether AQC-sys should use delegation is sensitive to the energy cost of running air purifier, the occupant's annoyance cost, and p_W. Since the costs associated with the air quality dominate the costs of energy annoyance, AQC-sys must always choose between running air purifier or delegating window-opening task to the occupant, and if the occupant refuses to open the windows, AQC-sys must run air purifier as a fallback. In the white region in Fig. 2, the expected cost of delegation is higher than the cost of running air purifier for `TAU`, because it is more likely that both annoyance cost and energy cost incur as a result of delegation. Thus, the better decision is to run air purifier – only energy cost incurs (shown as the flat region of the curved surface in Fig. 1). On the other hand, in the black region in Fig. 2, the expected cost of delegation is lower than the cost of running air purifier, because it is more likely that no cost incurs as a result of delegation. Thus, the better decision is to use delegation. The expected cost of delegation decreases as p_W increases, and as annoyance cost decreases except when $p_W = 1$. This analysis shows the average 32.93 % decrease in the expected total cost of system-human strategy compare to that of system-only strategy.

7 Related Work

Eskins and Sanders [5] introduce a definition of a cyber-human system (CHS) and the opportunity-willingness-capability (OWC) ontology for classifying CHS elements with respect to system tasks. This approach provides a structured and quantitative means of analyzing cyber security problems whose outcomes are influenced by human-system interactions, reflecting the probabilistic nature of human behavior.

There are some existing approaches for controller synthesis of systems with human operators. Li et al. [8] propose an approach for synthesizing human-in-the-loop discrete controller from temporal logic specification. They address the issue of devising a controller that is mostly automatic and requires only occasional human interaction for correct operation. Our work differs from theirs in that, while they focus on predicting the system's failure and notifying the human operator ahead of time, we focus on analyzing human factors to create cooperative strategy of the system and the human. Fu and Topcu [7] propose an approach for synthesizing shared autonomy policy that coordinates human operator and autonomous controller, by solving a multi-objective Markov decision

process with temporal logic specification. Their approach captures the evolution of the operator's cognitive state during control execution, and trades-off the human's effort and the system's performance level. While the trade-off analysis is similar to that of our work, our approach considers a more general notion of human factors, and thus it is appropriate for cyber-human systems of which humans are not necessarily have the system's operator.

Cámara et al. [6] propose a framework for analyzing the trade-offs of involving human operators in self-adaptation. Their work employs the OWC model to capture human factors, and uses model checking of SMG for analysis of how the human factors affect the outcome of adaptation, given a fixed adaptation strategy of the human operator. Our work has similar approach; however, we use strategy synthesis of SMG for devising cooperative strategy of the human and the system. Moreover, we emphasize on the interaction between the human and the system through delegation, and modeling of system-independent behavior of the human.

8 Conclusion

We explore the use of automated planning to design cooperation between cyber systems and humans in performing tasks, in which human participants have the role of actuators. We investigate how explicit modeling of human participants can be used in planning to generate cooperative strategies which best and appropriately utilize the humans. Specifically, our contributions are: (1) a formal framework for modeling cooperation between cyber systems and humans under uncertainty, with an explicit modeling of human participants based on opportunity-willingness-capabiity (OWC) ontology, and (2) a formalization of cyber system-human cooperative task planning as strategy synthesis of stochastic multiplayer game.

References

1. Chen, T., et al.: Automatic verification of competitive stochastic systems. Formal Methods Syst. Des. **43**(1), 61–92 (2013)
2. Kwiatkowska, M., Parker, D.: Automated verification and strategy synthesis for probabilistic systems. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 5–22. Springer, Heidelberg (2013)
3. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Computer Aided Verification. Springer, Berlin (2011)
4. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: a model checker for stochastic multi-player games. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 185–191. Springer, Heidelberg (2013)
5. Eskins, D., Sanders, W.H.: The multiple-asymmetric-utility system model: a framework for modeling cyber-human systems. In: 2011 Eighth International Conference on Quantitative Evaluation of Systems (QEST). IEEE (2011)

6. Cámara, J., Moreno, G.A., Garlan, D.: Reasoning about human participation in self-adaptive systems. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015) (2015)
7. Fu, J., Topcu, U.: Pareto efficiency in synthesizing shared autonomy policies with temporal logic constraints (2014). arXiv preprint arXiv:1412.6029
8. Li, W., Sadigh, D., Sastry, S.S., Seshia, S.A.: Synthesis for human-in-the-loop control systems. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014 (ETAPS). LNCS, vol. 8413, pp. 470–484. Springer, Heidelberg (2014)