A task-oriented service personalization scheme for smart environments using reinforcement learning

Conference Paper · March 2016

DOI: 10.1109/PERCOMW.2016.7457110

Bjorn Tegelund
Korea Advanced Institute of Science and Techn…
3 PUBLICATIONS **5** CITATIONS

Heesuk Son
Korea Advanced Institute of Science and Techn…
10 PUBLICATIONS **8** CITATIONS

A Task-oriented Service Personalization Scheme for Smart Environments Using Reinforcement Learning

Bjorn Tegelund, Heesuk Son, Dongman Lee
School of Computer Science, KAIST
Daejeon, S. Korea
{bjorn, heesuk.son, thodlee}@kaist.ac.kr

***Abstract*—Users want IoT environments to provide them with personalized support. These environments therefore need to be able to learn user preferences, such as what temperature the room should be or which lights should be turned on. We propose a novel system that separates the tasks of learning a user's preferences and realizing them within the environment. The system is able to capture user preferences by reinterpreting the problem as an optimization problem and applying inverse reinforcement learning to it. The system is shown to be able to accurately extract simple preferences given a small number of user demonstrations. These preferences are then realized by actuates devices running reinforcement learning-based agents to provide an environment consistent with the learnt preferences, also in situations not included in any user demonstration.**

I. Introduction

Ambient systems are becoming more available and diversified. These systems are tasked with providing services to users while taking the current situation as well as given and derived knowledge into account. There are however many challenges remaining before we can provide users with smart environments that fulfill the vision of ambient intelligence, such as how to seamlessly personalize services. Manufacturers try to predict as many service use cases as possible, but it is impossible to account for every type of user and every context. Requiring users to manually configure everything themselves is impractical, as even technologically passionate users find such systems cumbersome and overly time-consuming [1]. Systems therefore need to be able to figure out users' individual requirements and adapt accordingly at runtime, without explicit user intervention.

Recent works related to ambient intelligence such as [2] and [3] are focused on mimicking user behavior. Whenever a user performs an action, the context in which the action is performed is recorded and associated with the action, producing a mapping from context to action without any user configuration. Personalization is then done by capturing the specific context in which the user performs an action. However, when extending this approach to multiple devices to create a smart environment, shortcomings become apparent, namely a lack of cooperation brought on by the absence of shared goals and inflexibility caused by the need for each device to learn everything from scratch. As each device is only concerned with the creation of its own context-to-action mapping, how each action affects the environment is largely ignored.

We argue that creating such a mapping for each device is a fundamentally flawed model. A user may take one or multiple actions over a period of time to, for example, lower the room temperature, but such actions are all captured independently. Instead, the system should collectively capture those actions with respect to a user's goal and intended task. Furthermore, each device requires a user to demonstrate what to do in each undesirable context, thus requiring large amounts of training data. By interpreting the personalization of a smart environment as the provision of a preferred environment, a common goal that all devices share is created.

In this paper, we propose a task-based personalization scheme using reinforcement learning. The scheme consists of two parts: (1) preference capturing *"what is the user's preferred environment for the current task?"* and (2) preference realization *"how can each smart device, at this moment, contribute to realizing that environment?"* This is done by reinterpreting the provision of the preferred environment for the current task as an optimization problem, where preference capturing corresponds to deriving a function describing the preferability of a context, and preference realization corresponds to actuating devices to maximize the function value. Capturing this function becomes possible by collecting user demonstrations and applying inverse reinforcement learning to them. This is then used by reinforcement learning agents in devices to calculate which action gives the largest environment preferability gain. The system is shown to be responsive, making decisions within a few milliseconds, and able to capture and realize preferences given a small number of user demonstrations as long as the data does not contain any contradictions.

The rest of this paper is organized as follows: related works are discussed in section II. The design considerations and resulting system design are listed in sections III and IV. Section V presents the evaluation setup and results. Section VI concludes the paper and proposes future work.

II. Related Work

Lee et al. [3] propose a context-aware system to control a smart home using sensors and connected devices. They use three different types of inferences to derive knowledge about what users are currently doing and what services should be provided. Their system is able to learn when to provide services by looking at when a user manually actuates a device using case-based inference. However, this is solely based on the day and time, along with what the inferred behavior is, e.g. “getting up in the morning”. Their system also requires the user to perform certain configuration manually, such as what confidence level is required for each service to be provided.

Guivarch et al. [2] describe an adaptive multi-agent architecture for controlling smart devices based on user behavior. Learning is done without explicit user intervention by looking at the current context and associating that with the action taken by the user. If the confidence level for an action is high enough, it is taken automatically, thereby providing support to users.

These two works try to mimic the action the user takes without a concept of user preference. What a user wants is not a device that learns to take a specific action in a specific context, but a context that is always in accordance with their preferences. The main disadvantage of the first approach is that each device has to learn from user demonstration which action to take whenever the context is not preferable, instead of learning what context is preferable and acting to realize it.

Sasidharan et al. [4] implement a cognitive management framework that is used both to control appliances in a home and monitor the user’s status. In their system, a neural network is used to learn which temperature the home should be at which time of day by observing user actions. This information is then used to actuate devices and realize the preference with high accuracy. Their system is able to achieve a similar concept of preference as that presented in this paper, but only for a single variable. For this approach to extend to multiple contexts and devices, a large number of neural networks would be required along with a large amount of training data.

III. Design Considerations

The smart environment is assumed to consist of devices with computation, storage, and networking capabilities, running a context-aware middleware that provides common services such as context modeling, context dissemination, and device discovery. Given these assumptions, the proposed scheme has been designed with the following considerations in mind:

A. Preference Organization

Preferences are context-dependent. Preferences vary depending on a number of factors, such as what the user is doing, or who they are with. Consider the scenarios of watching a movie versus reading a book. The user may prefer dimmed lighting in the former case and full lighting in the latter. Therefore, selecting an appropriate preference organization scheme is important and affects the way preferences are captured.

The main goal of our system is to provide an optimal environment for a user to complete their current task in, so an organization of preferences based on the user’s current task is chosen. A task is, in this case, a sufficiently high-level description of what the user is currently doing, as described in [5]. The task is used as a label to organize preferences and is assumed to be provided as part of the environment context.

B. Preference Capturing

Before a person engages in a task, he or she commonly begins by modifying the environment to better fit the planned task and his or her preferences. For example, when watching a movie, a user might dim the lights, turn on the surround sound system, switch the TV input to the DVD-player, among other things. As these actions are done before the actual movie watching begins, this process could be thought of as the user optimizing the environment to better fit the planned task.

A machine learning algorithm is employed to learn the user’s preferences by observing this optimization process. When selecting which algorithm to use for this, the main criterion in is the amount of data required to model preferences accurately enough to quality support to the user. This correlates to the time required until the system is useful, and must therefore be minimized. The output of the algorithm should represent the user’s preferences for the current task.

Given the above, we believe that inverse reinforcement learning [6] is a good fit for the preference capturing problem. The main advantage of inverse reinforcement learning is that it is able to directly take the observed actions of the user as input data. Furthermore, given the strong assumption that each action slightly improves the environment, the algorithm is able to recover an accurate preference function with a small amount of training data compared to other algorithms [6]. The disadvantage of this approach is that the effect of each action on the environment has to be known in advance. This is easy to specify for simple things like doors, but more complicated in the case of lamps (as the current room brightness affects the measured change in brightness) or window blinds (which depend on the brightness outside). Measuring the effects of the actions where possible and adding additional sensors as needed should be done to reduce the complexity of this.

C. Preference Realization

To facilitate the realization of a user’s preferences, the devices in the environment need a method to derive how to further optimize the environment. To enable weighting different kinds of preferences against each other, the method should be able to take multiple objectives into account when performing decision. For example, environmental preferences, e.g. temperature, can be weighed against device preferences, which describe how often the user prefers to use a specific device to change the environment in a certain way. Note that resolving conflicting preferences between users is out of scope for this work.

Reinforcement learning is chosen as the algorithm to achieve this, as it is capable of multi-objective decision-making and can make decisions in $O(n)$ of the number of actions available in the current state, which is a small number [7]. The main advantage of this approach is that it can directly use the preference function generated by the inverse reinforcement learning algorithm as a reward function to train an agent. Since the agent acts to maximize environment preferability each time the state is changed, the system has a reactive quality, which is desirable in intelligent systems.

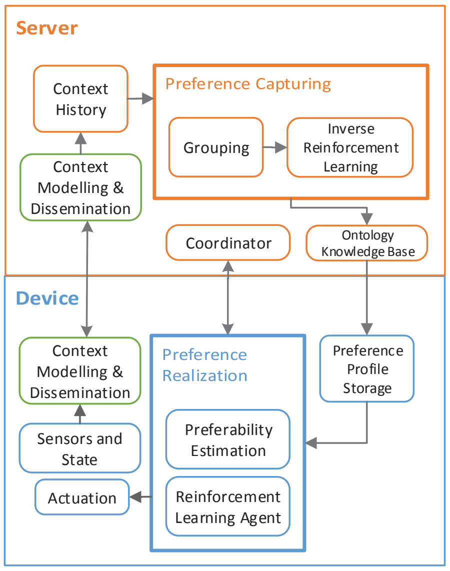

Fig. 1. System Architecture - boxes with green outlines are assumed to be part of the middleware

IV. SYSTEM DESIGN

In this section, we describe our system design based on the previously presented design considerations. To summarize, context is shared between devices and the server using functionality assumed to be present in the middleware. The changes in context and their reasons are stored in a context history and filtered to extract preferences for each user and task combination. These preferences along with other information about the user and environment are stored in a knowledge base. For each user, a preference profile is generated, consisting of all captured preferences for that user, and sent to each device for agent generation. Agents elect their most preferable action along with an estimated preferability increase to the coordinator, which picks the best action to perform. That action is then actuated by the device, triggering another change in context which is used to poll devices for their next actions.

The architecture is explained in further detail by following the flow of data through Figure 1 with the movie watching example previously mentioned in section III-B. In this case, we assume an simplified environment consisting of a TV (with states “on” and “off”), a lamp (with similar states), a set of window blinds (that can be “raised” or “lowered”) and a sensor for capturing brightness, where the user turns off the lamp and closes the window blinds before watching a movie.

A. Context History

Changes in context within the environment and collected and condensed into a context history. The context is a description of the current environment, and can be visualized as a table with columns for all environmental variables as well as devices states. The context may be changed by actions (e.g. the user turns on the TV, or the system turns on a lamp) or they may occur naturally (e.g. room brightness changing due to light coming in through windows).

B. Preference Capturing

After the context history has been grouped based on user and task, it is separated into episodes, each denoting an instance of a task being performed. All episodes for each user and task combination are gathered and preference capturing is performed. This is done using an inverse reinforcement learning algorithm based on [6].

To apply this algorithm, the smart environment domain must be mapped to a Markov decision process (MDP) consisting of a tuple (S, A, P, R). This is done by mapping S (the state space) to include all possible contexts within the environment, A (the action space) to the set of device actions, e.g. turn off lamp or raise window blinds, and P (a probability function $S \times A \times S \rightarrow [0, 1]$) to be 1 in all cases, as it is assumed that actions always succeed with predictable results.

R is the reward function, which is recovered by the inverse reinforcement learning algorithm. Let R : *context* $\rightarrow \mathbb{R}$ to be a function that is known to the user, which describes how close the current context is to the user’s preferences for the current task. This is considered to be reasonable, as the user knows their own preferences and can easily judge which of two potential environments is preferable for the task at hand. This function can then be recovered by the algorithm by inputting episodes of the user optimizing the function, e.g. turning off the light and lowering the window blinds in preparation for watching a movie. When extracting R, it is assigned the form $R(s) = w^T\phi$, $s \in S$, where w is a weight vector that is generated by the algorithm and ϕ is a feature vector, consisting of the features that should be considered when calculating the reward, e.g. if the TV is on or not.

While there are many ways that preferences could be expressed using a linear combination, ϕ has to be designed to be general enough to be able to express all kinds of environmental and device preferences, as well as precise enough to allow for exact realization of preferences. The chosen design is to set ϕ to be a binary vector representing all environmental variables along with device states. To transform discrete variables, such as device states, to this format, they are modeled as vectors that are as long as the domain of the variable. For example, given a lamp with the states “on” and “off”, a feature vector $[on, off]$ is produced. A context snapshot can then be represented by a feature vector by comparing each feature to the current context and setting them to 1 or 0 depending on if they match or not. Given the previous lamp feature vector, a context snapshot where the lamp is on would yield $[1, 0]$. Continuous variables, such as room brightness, are handled in a similar way by first performing normalization before splitting this domain into a chosen number of intervals, thereby discretizing them. Depending on how many intervals are chosen, the granularity of the preferences captured can be increased at the cost of longer runtime. ϕ then becomes the concatenation of all variables’ feature vectors. In the example, brightness might take values $0 - 300$, which

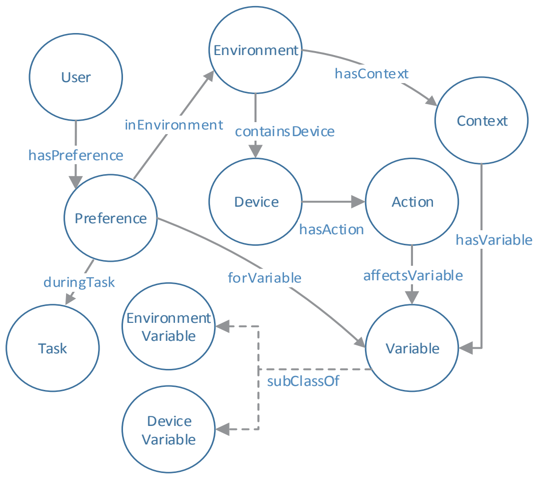

Fig. 2. Part of the ontology scheme used in the knowledge base for representing smart environments and user preferences

could after normalization be discretized into three intervals: $[0-0.33\,(low), 0.33-0.66\,(medium), 0.66-1\,(high)]$. Given a context where the TV is off, the window blinds are raised, and the lamp is on, producing a high brightness, this would be mapped to a feature vector in the following way:

$$[\overbrace{on, off}^{TV}, \overbrace{on, off}^{Lamp}, \overbrace{lowered, raised}^{Window\ blinds}, \overbrace{low, medium, high}^{Brightness}]$$
$$\Rightarrow \phi = [0, \underline{1}, \underline{1}, 0, 0, \underline{1}, 0, 0, \underline{1}]$$

The corresponding weight that is calculated by the algorithm for each feature represents how preferable that feature is, so a large weight for the first feature would imply a strong preference for having the TV on. As such, $w^T\phi \in \mathbb{R}$ and describes the preferability score of the context. w is then stored in the knowledge base.

C. Knowledge Base

The knowledge base is composed of an ontology which provides a basic vocabulary for different devices to reason about their environment and facilitate communication largely based on [8]. It contains descriptions of devices and their actions, as well as references to functions for calculating how an environment changes when an action is applied and for translating contexts into feature vectors. Basic user information is also contained in this ontology. As can be seen from Figure 2, the learned preferences are associated with a specific user, task, and location. Different locations are thought of as different environments, and user preferences may vary among them, e.g. the user might prefer no lighting when watching TV in the bedroom, but strong lighting when watching TV while cooking in the kitchen.

D. Preference Realization

All estimated preference functions associated with a user forms a preference profile, where a specific function is chosen based on the current task. Preference profiles are all distributed to all devices. Devices train one reinforcement learning agent for each preference function by using it as the reward function when running SARSA(λ) [9], which is a kind of reinforcement learning algorithm. The MDP mentioned in section IV-B is modified for each device; S only consists of contexts with environmental variables relevant to the device, and A becomes the device's actions. For the lamp, to which only brightness is relevant, $A = [turn\ on, turn\ off]$ and $S = [low, medium, high] \times [on, off]$. S is built automatically by retrieving information about what environmental variables are affected by the device's actions from the knowledge base. This can be done as, for example, the action that should be taken by the lamp is irrelevant of the environment's humidity. A benefit of narrowing down S is reducing the number of states that the agent has to visit to $\Pi_{variable \in s} Dom(variable)$, thereby reducing training time.

As the behavior of a reinforcement learning agent is determined by its reward function and algorithm parameters, these must be designed to enable and strengthen the kinds of behaviors desired from the agent. Consider the Q-learning formula [9] used by the SARSA(λ) algorithm to enable agents to learn new information:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

where $Q(s_t, a_t)$ is the expected reward for following the best sequence of actions after taking action a_t in state (or in terms of this paper, context) s_t, and r_{t+1} is the estimated reward received for taking a_t in s_t. The learning rate α (how aggressively the Q values are updated) and discount factor γ (how far into the future the agent considers actions) need to be adapted to this problem. Since most agents, such as that for the lamp or other simple appliances, only need to take one or two actions, a low $\gamma \in [0.0, 0.3]$ is chosen. A large $\alpha \in [0.8, 1.0]$ is conventional as rewards will not change between iterations of agent training. The initial expected reward $Q_0 = 1.0$ is chosen to encourage exploration of different states during the learning phase and ensure all states have been explored. Rewards for environmental and device preferences are separated, and a higher weight is given to the environmental reward to encourage the agents to first and foremost optimize the environment, yielding

$$r_{t+1} = \beta_e r_{t+1}(environment) + \beta_s r_{t+1}(device\ state)$$

where $\beta_e > \beta_s$. β_e and β_s can then be adjusted as needed for a specific environment. The preferred device will then be chosen to take the action which improves the environment, as it will have a higher reward than any other device that may be able to perform the same action.

While many systems employ a default negative reward to reinforcement learning systems to force agents to act, this is not the case in our system. This is chosen as the negative effect on user experience when not giving support when it is needed is lower than that of giving it when it is not [1].

After agent training is done, context snapshots can be fed into the agent and the most preferable action in the context is computed and reported together with a preferability score to the coordinator.

E. Coordination

The coordinator's role is to ensure that devices perform actions, one at a time, that increase the preferability of the environment by a meaningful amount. Each time the context changes, devices are polled to see if there are any actions that should be taken. After each device has selected their most preferable action, the coordinator selects the one with the highest preferability score to be actuated. Once the action has been performed and taken its effect on the environment, the coordinator polls the devices again. As the focus of this paper is service personalization, a more advanced coordination scheme is deferred to future work.

V. Evaluation

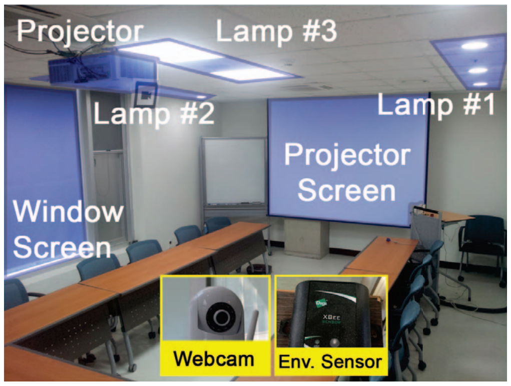

Fig. 3. Testbed setup showing devices and sensors

A. Experiment Setup

In evaluating the proposed scheme, the primary metrics of interest are the accuracy in capturing user preferences and the ability to actuate devices to provide an environment in accordance with those preferences. Evaluation is done by obtaining these metrics for three different tasks that were regularly performed at similar times over a two month period in our smart meeting room testbed. The tasks chosen are "movie watching", "student discussion", and "lab meeting". For the "lab meeting" and "discussion" tasks, data from multiple groups were obtained. Each group is treated as a single person by the system to remove the need to handle colliding preferences. An analysis of system response time, preference capturing time, and agent training times is also included.

The devices included in the testbed are shown in Figure 3. The webcam is used to capture ground truth. All devices are controlled by Raspberry Pi 2 Model B boards, and run a custom IoT middleware described in [8]. Note that only the middleware's context modeling and dissemination modules are used in this paper.

For each task and group combination, a set of five episodes are captured and processed to form a context history to base learning upon. Additional webcam footage is also included for each episode. Context and order of actions often vary between episodes for the same task and context, thereby introducing variance between episode, as would be expected in a real-world application. An example episode of the actions taken during a lab meeting for group #1 is: "Projector On", "Lamp #1 On", "Lamp #2 On", "Lamp #3 On", "Projector Screen Down". Through manual analysis of these instances, the characteristics of each task are extracted and summarized in the first six columns of Table I.

For all experiments, agents are trained with parameters $\alpha = 0.8$, $\gamma = 0.1$ for reasons stated in section IV-D, and $\beta_e = 5, \beta_s = 1$ are used to highly prioritize optimizing the environment. The agents are trained for 3000 iterations each to ensure that they have explored as many contexts as possible. The inverse reinforcement learning and reinforcement learning algorithms are available through the BURLAP library [10].

B. Experiment - Preference Capturing

The preference capturing mechanism is found to be able to derive the expected preferences in accordance with Table I. The preferences are extracted by selecting the value corresponding to the largest weight for each feature. There is some variation between the episodes in the case of discussion for group #3, which leads to difficulty in deriving the preference related to the projector screen. In two of the five episodes, the projector screen is already lowered when the task begun, while in the other three it is not. The system cannot in the case of this contradiction derive whether the users prefer to have the projector screen lowered or raised, so the same weights are calculated for both features. The algorithm is therefore able to derive the correct preference for 34 out of the 35 preferences to be learnt (seven preferences for each of the five tasks), giving it an accuracy of 97%. Upon further analysis, it is found that if the training data related to a variable contains 10% contradictions, the algorithm fails in 20% of cases to compute any preference for this variable.

C. Experiment - Preference Realization

To evaluate the proposed scheme's ability to realize preferences, a "default" context where the lamps and projector are turned off and the projector screen and window blinds are raised is fed to the system for each set of episodes. The actions taken by the system and their order are included in the last column of Table I. As can be seen from the table, the system is able to achieve the preferred context for all tasks.

When a context completely opposite to the preferences of the discussion task for group #3 is given (all lamps on, window blinds and projector screen down, projector on), the system is able to actuate devices to restore the preferred environment, with the exception of the projector screen for reasons stated in the previous subsection. Worth noting is that this is a context which is not contained in training data. The system is able to take the most vital actions to the environment first, i.e. raising the window blinds and turning off the lamps before turning off the projector.

TABLE I
CHARACTERISTICS OF EACH GROUP & TASK AND SYSTEM ACTIONS TAKEN

Group	Task	Time	Participants	Brightness	Used Devices**	System Actions in Default Context
1	Lab Meeting	8 PM	≥ 10	140	Lamps #1-3, Projector, Screen	Lamp #1 On, Lamp #3 On, Lamp #2 On, Screen Down, Projector On
2	Lab Meeting	11 AM	$4-5$	300*	Projector, Screen	Projector On, Screen Down
3	Discussion	2 PM	$2-5$	300*	None	None
4	Discussion	9 PM	$2-3$	130	Lamps #1-2	Lamp #2 On, Lamp #1 On
5	Movie Watching	4 PM	$1-2$	0	Projector, Screen, Blinds	Blinds Down, Screen Down, Projector On

* Outside light coming in through the window, ** The projector screen is shortened to "screen" and the window blinds to "blinds"

D. Performance

The performance metrics of interest are the preference capturing time, agent training time, and decision-making time. Preference capturing is done on a server with an Intel i5 2.90GHz processor and 4GB of RAM, which is deemed a reasonable hardware configuration to serve as a central server for all smart environments within a building. When calculating a preference function for a task, the algorithm either runs until convergence or at most 30 iterations, which takes 26 seconds on average.

The agents are trained on Raspberry Pi 2 Model B boards, with 900MHz quad-core ARM processors and 1GB of memory. Training takes on average 130s for lamps and window blinds, and 49s for the projector and projector screen. The average decision-making time for all devices is 5ms.

E. Discussion

From the results, it is apparent that the proposed scheme is able to capture preferences independent of the order they are performed as long as there are not any contradictions within the data. While a small amount of contradiction can be handled if the amount of data is large enough, it can easily cause problems. If there is low variance between episodes, the algorithm is able to extract preferences accurately even though the amount of training data is small. The realization of these preferences is reliable, accurate, and can be done in contexts not included in training data. This is because the system does not rely on direct user demonstration of which action to take in an undesirable context. The proposed scheme is capable of going from the context history to a set of trained agents for a specific task within 3 minutes, as agents are distributed and trained in parallel. As most tasks do not happen more than a few times per day, calculations are fast enough to be redone after every episode, meaning that the proposed scheme appears reactive even though all learning and training is done in an offline manner. Decision-making times within the agents are fast, suggesting that the proposed scheme will respond quickly even in environments with large numbers of devices.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present a personalization scheme for capturing and realizing user preferences in a smart environment. The scheme uses a previously unseen approach based on inverse reinforcement learning to capture preferences. Preferences are captured accurately and independently of action order for simple tasks as long as there are no contradictions within the data. The proposed scheme is also able to actuate devices to realize an environment in accordance with user preferences even in previously unseen contexts. The scheme has also been shown to be performant when running on hardware typically found in modern smart devices. We hope that the findings presented within this paper can open up a new avenue of research focused on further applications of inverse reinforcement learning within smart environments.

As future work, we plan to extend the scheme to perform preference conflict resolution between users. A more extensive user study in various smart environments is also planned, where transfer of preferences between environments will be investigated.

ACKNOWLEDGEMENT

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korean government (MSIP) (R0184-15-1037, Development of Data Mining Core Technologies for Real-time Intelligent Information Recommendation in Smart Spaces).

REFERENCES

[1] S. Makonin, L. Bartram, and F. Popowich, "A smarter smart home: Case studies of ambient intelligence," *IEEE Pervasive Computing*, no. 1, pp. 58–66, 2013.

[2] V. Guivarch, V. Camps, and A. Péninou, "Context awareness in ambient systems by an adaptive multi-agent approach," in *Ambient intelligence*. Springer, 2012, pp. 129–144.

[3] J. H. Lee, H. Lee, M. J. Kim, X. Wang, and P. E. Love, "Context-aware inference in ubiquitous residential environments," *Computers in Industry*, vol. 65, no. 1, pp. 148–157, 2014.

[4] S. Sasidharan, A. Somov, A. R. Biswas, and R. Giaffreda, "Cognitive management framework for internet of things:a prototype implementation," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 538–543.

[5] T. Naganuma and S. Kurakake, "Task knowledge based retrieval for service relevant to mobile users activity," in *The Semantic Web–ISWC 2005*. Springer, 2005, pp. 959–973.

[6] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[7] M. Amoui, M. Salehie, S. Mirarab, and L. Tahvildari, "Adaptive action selection in autonomic software using reinforcement learning," in *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*. IEEE, 2008, pp. 175–181.

[8] H. Son, B. Tegelund, T. Kim, D. Lee, S. J. Hyun, J. Lim, and H. Lee, "A distributed middleware for a smart home with autonomous appliances," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 2. IEEE, 2015, pp. 23–32.

[9] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994.

[10] J. MacGlashan. (2015) Burlap library. [Online]. Available: http://burlap.cs.brown.edu/index.html