Domain-independent User Simulation with Transformers for Task-oriented Dialogue Systems

Hsien-chin Lin[1], Nurul Lubis[1], Songbo Hu[2], Carel van Niekerk[1],
Christian Geishauser[1], Michael Heck[1], Shutong Feng[1], and Milica Gašić[1]

[1]Heinrich Heine University Dusseldorf, Germany
[2]Department of Computer Science and Technology, University of Cambridge, UK
[1]{linh,lubis,niekerk,geishaus,heckmi,shutong.feng,gasic}@hhu.de
[2]sh2091@cam.ac.uk

Abstract

Dialogue policy optimisation via reinforcement learning requires a large number of training interactions, which makes learning with real users time consuming and expensive. Many set-ups therefore rely on a user simulator instead of humans. These user simulators have their own problems. While hand-coded, rule-based user simulators have been shown to be sufficient in small, simple domains, for complex domains the number of rules quickly becomes intractable. State-of-the-art data-driven user simulators, on the other hand, are still domain-dependent. This means that adaptation to each new domain requires redesigning and retraining. In this work, we propose a domain-independent transformer-based user simulator (TUS). The structure of our TUS is not tied to a specific domain, enabling domain generalisation and learning of cross-domain user behaviour from data. We compare TUS with the state of the art using automatic as well as human evaluations. TUS can compete with rule-based user simulators on pre-defined domains and is able to generalise to unseen domains in a zero-shot fashion.

1 Introduction

Task-oriented dialogue systems are designed to help users accomplish specific goals within a particular task such as hotel booking or finding a flight. Solving this problem typically requires tracking and planning (Young, 2002). In tracking, the system keeps track of information about the user goal from the beginning of the dialogue until the current dialogue turn. In planning, the dialogue policy makes decisions at each turn to maximise future rewards at the end of the dialogue (Levin and Pieraccini, 1997). The system typically needs thousands of interactions to train a usable policy (Schatzmann et al., 2007; Pietquin et al., 2011; Li et al., 2016; Shi et al., 2019). The amount of interactions required makes learning from real users time-consuming and costly. It is therefore appealing to automatically generate a large number of dialogues with a user simulator (US)[1](Eckert et al., 1997).

Rule-based USs are interpretable and have shown success when applied in small, simple domains. However, expert knowledge is required to design their rules and the number of rules needed for complex domains quickly becomes intractable (Schatzmann et al., 2007). In addition, handcrafted rules are unable to capture human behaviour to its fullest extent, leading to sub-optimal performance when interacting with real users (Schatzmann et al., 2006).

Data-driven USs on the other hand can learn user behaviour directly from a corpus. However, they are still domain-dependent. This means that in order to accommodate an unseen domain one needs to collect and annotate a new dataset, and retrain or even re-engineer the simulator.

We propose a transformer-based domain-independent user simulator (TUS). Unlike existing data-driven simulators, we design the feature representation to be domain-independent, allowing the simulator to easily generalise to new domains without modifying or retraining the model. We utilise a transformer architecture (Vaswani et al., 2017) so that the input sequence can have a variable length and dynamic order. The dynamic order takes into account the user's priorities and the varying input length enables the US to incorporate system actions in a seamless manner. TUS predicts the value of each slot and the domains of the current turn, allowing the model to optimise its performance in multiple granularities. By disentangling the user behaviour from the domains, TUS can learn a more general user policy to train the dialogue policy.

[1]There are approaches that attempt to learn a dialogue policy from direct interaction with humans (Gašić et al., 2011). Even then, USs are essential for development and evaluation.

We compare policies trained with our TUS to policies trained with other USs through indirect and direct evaluation as well as human evaluation. The results show that policies trained with TUS outperform those that are trained with another data-driven US and are on par with policies trained with the agenda-based US (ABUS). Moreover, the policy generalises better when evaluated with a different US. Automatic and human evaluations on our zero-shot study show that leave-one-domain-out TUS is able to generalise to unseen domains while maintaining a comparable performance to ABUS and TUS trained on the full training data.

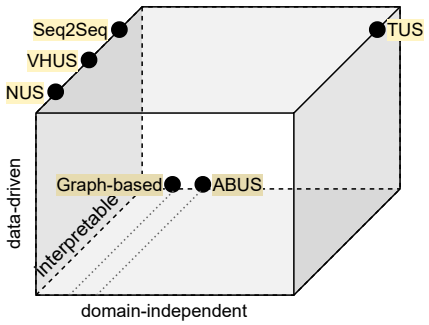

Figure 1: The difference between USs. We compare to which extent a model is data-driven, domain-independent and interpretable.

2 Related Work

The quality of a US has a significant impact on the performance of a reinforcement-learning based task-oriented dialogue system (Schatzmann et al., 2005). One of the early models include an N-gram user simulator proposed by Eckert et al. (1997). It uses a 2-gram model $P(a_u|a_m)$ to predict the user action a_u according to the system action a_m. Since it only has access to the latest system action, its behaviour can be illogical if the goal changes. Therefore, models which can take into account a given user goal were introduced (Georgila et al., 2006; Eshky et al., 2012). The Bayesian model of Daubigney et al. (2012) predicts the user action based on the user goal, and hidden Markov models are used to model the user and the system behaviour (Cuayáhuitl et al., 2005). The graph-based US of Scheffler and Young (2002) combines all possible dialogue paths in a graph. It can generate reasonable and consistent behaviour, but is impractical to implement, since extensive domain knowledge is required.

The agenda-based user simulator (ABUS) (Schatzmann et al., 2007) models the user state as a stack-like agenda, ordered according to the priority of the user actions. The probabilities of updating the agenda and choosing user actions are set manually or learned from data (Keizer et al., 2010). Still, the stacking and popping rules are domain-dependent and need to be designed carefully.

To build a data-driven model, the sequence-to-sequence (Seq2Seq) model structure is widely used. El Asri et al. (2016) propose a Seq2Seq semantic level US with an encoder-decoder structure. Each turn is fed into the encoder recurrent neural network (RNN) and embedded as a context vector. Then this context vector is passed to the decoder RNN to generate user actions. To add new domains, it is necessary to modify the domain-dependent feature representation and retrain the model.

Instead of generating semantic level output, the neural user simulator (NUS) by Kreyssig et al. (2018) generates responses in natural language, thus requiring less labeling, at the expense of interpretability. However, its feature representation is still domain-dependent.

A variational hierarchical Seq2Seq user simulator (VHUS) is proposed by Gür et al. (2018). Instead of designing dialogue history features, the model encodes the user goal and system actions with a vector using an RNN, which alleviates the need of heavy feature engineering. However, the inputs are represented as one-hot encodings, which are also dependent on the ontology. In addition, the output generator is not constrained by the ontology in any way, so it can generate impossible actions.

As shown in Fig. 1, ABUS and graph-based models are domain-dependent and require significant design efforts. Data-driven models such as Seq2Seq, NUS, and VHUS can learn from data, but are constrained by the underlying domain. NUS generates natural language responses, which requires less labeling, but comes with reduced interpretability.

Shi et al. (2019) compared different ways to build a US and indicated that the data-driven models suffer from bias in the corpus. If some actions are rare in the corpus, the model cannot capture them. Thus, the dialogue policy cannot explore all possible paths during training with the data-driven USs. It is important to learn more general human behaviour to reduce the impact of the corpus bias.

3 Problem Description

Task-oriented dialogue systems are defined by a given *ontology*, which specifies the concepts that the system can handle. The ontology can include multiple *domains*. In each domain, there are *informable slots*, which are the attributes that users can assign *values* to, and *requestable slots*, which are the attributes that users can query. For example, in Fig. 2 the user goal has two domains, "hotel" and "restaurant". The slot `Area` is an informable slot with the value `North` in domain "hotel" and `Addr` is a requestable slot in domain "restaurant". The *system state* records the slots and values mentioned in the dialogue history. A US for task-oriented dialogue systems needs to provide coherent responses according to a given user goal $G = \{domain_1 : [(slot_1, value_1), (slot_2, value_2), \dots], \dots\}$. The domains, slots and values are selected from the ontology.

The *user action* is composed of user intents, domains, slots, and values. We consider user intents that appear in the MultiWOZ dataset (Budzianowski et al., 2018). It is of course possible to consider arbitrary intents within the same model architecture, as long as they are defined a priori[2]. The two possible user intents we consider are *Inform* and *Request*. With *Inform*, the user can provide information, correct the system or confirm the system's recommendations. When a user goal cannot be fulfilled, the user can also randomly select a value from the ontology and change the goal. With *Request*, the user can request information about certain slots.

The *system action* is similar to the user action, but there exist more (system) intents. For example, the system can provide suggestions to users with the intent *Recommendation* and make reservations for users with the intent *Book*. More system intents can be found in Appendix A.

We view user simulation in a task-oriented dialogue as a sequence-to-sequence problem. For each turn t, we extract the input feature vectors V^t of the input list of slots $S^t = [s_1, s_2, \dots]$, which is composed of the slots from the user goal and the system action. The output sequence $O^t = [o_1^t, o_2^t, \dots]$ is then generated by the model, where o_i^t shows how the value for slot s_i is obtained. The input feature representation and the output target should be domain-independent in order to generalise to unseen domains without redesigning and retraining. More details can be found in Sec. 4.

[2]We note that intents are not normally dependent on the domain but rather on the kind of dialogue that is being modeled, e.g. task-oriented or chit-chat.

User Goal
`Info:` Hotel-Area=North, Rest-Area=North
`Reqt:` Hotel-Name, Rest-Addr
Conversation
`Turn 0`
`USR:` I want to find a hotel in the north and a nearby restaurant.
`Inform(Hotel-Area=North, Rest-Area=North)`
`SYS:` There are some good hotels in the south. Which price range do you prefer? Would you mind providing more information?
`Recom(Hotel-Area=South), Request(Hotel-Price), general-reqmore()`
`Turn 1`
`USR:` No, I want one in the north and I don't care about the price range.
`Inform(Hotel-Area=North, Hotel-Price=dontcare)`

Figure 2: An example dialogue with a multi-domain goal.

By working on the semantic level during training, we retain interpretability. To interact with real users during human evaluation, we rely on template-based natural language generation to convert the semantic-level actions into utterances, as language generation is out of the scope of this work.

4 Transformer-based Domain-independent User Simulator

The TUS model structure is shown in Fig. 3. For each turn t, the list of input feature vectors $V^t = [v_1^t, v_2^t \dots, v_{n_t}^t]$ is generated based on the system actions and the user goal, where v_i^t is the feature vector of slot s_i and n_t is the length of the input list in turn t, V^t. We explain the feature representation in detail in Sec. 4.1. Inspired by ABUS, which models the user state as a stack-like agenda, the length of input list n_t at each turn t varies by taking into account slots mentioned in the system's action. For example, in Fig. 3 the input list V^0 only contains the slots in the user goal at the first turn. Then the system mentions a slot not in the user goal, `Hotel-Price`. So in turn 1 the length of input list V^1 is $n_1 = n_0 + 1$ because one slot is inserted into the input list V^1. The whole input sequence to the model is $V_{input} = [v_{CLS}, v_1^t, \dots, v_{SEP}, v_1^{t-1}, \dots, v_{SEP}]$, where v_{CLS} is the representation of `[CLS]` and v_{SEP} is the representation of `[SEP]`.

The user policy network is a transformer (Vaswani et al., 2017; Devlin et al., 2019). We choose this structure because transformers are able to handle input sequences of arbitrary lengths and to capture the relationship between slots thanks

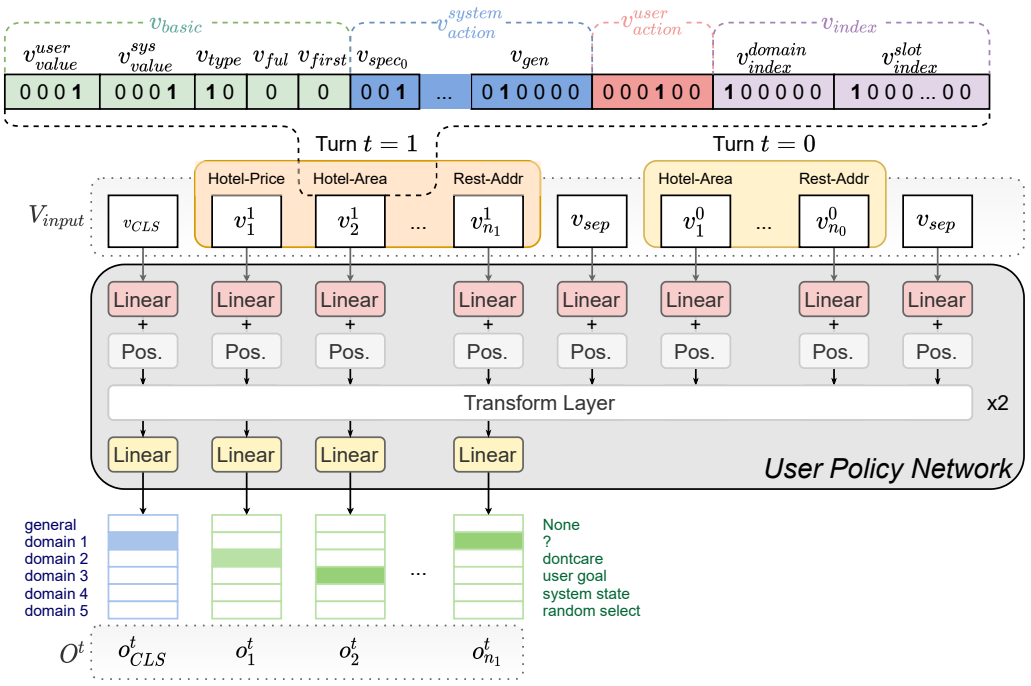

Figure 3: The TUS model structure. The input list starts with a special token, `[CLS]`, and comprises slot lists from previous turns. The slot lists from each turn are separated by a token, `[SEP]`. The model predicts an output vector for each slot in the last turn. Note that the order of slots in each turn is independent from each other. The output for `[CLS]` represents which domains should be selected in the current turn. The user goal and dialogue history are shown in Fig. 2 and here we give the example of the input feature v_i for slot `Hotel-Area`.

to self-attention. The model structure includes a linear layer and position encoding for inputs, two transformer layers, and one linear layer for outputs.

The output list $O^t = [o_1^t, \ldots, o_{n_t}^t]$ consists of one-hot vectors o_i^t which determine the values of the slots s_i at turn t. The dimensions of $o_i^t \in \{0, 1\}^6$ correspond to "none", "don't care", "?", "from the user goal", "from the system state", or "randomly selected". More precisely, "none" means that this slot is not mentioned in this turn, "don't care" signifies that the US does not care about this slot, "?" means the US wants to request information about this slot, "from user goal" implies that the value is the same as in the user goal, "from system state" means that the value is as mentioned by the system, and lastly "randomly selected" indicates that the US wants to change its goal by randomly selecting a value from the ontology.

The loss function for slots measures the difference between the predicted output O^t and the target Y^t at each turn t from the dataset as computed by cross entropy (CE), i.e.,

$$loss_{slots} = \frac{1}{n_t} \sum_{i=1}^{n_t} \mathrm{CE}(o_i^t, y_i^t), \qquad (1)$$

where n_t is the number of slots in the input list, o_i^t is the output, and y_i^t is the target of slot s_i in turn t.

4.1 Domain-independent Input Features

We design the input feature representation v_i^t of each slot s_i in turn t consisting of a set of sub-vectors, all of which are domain-independent. For better readability, we drop the slot index i and the turn index t, i.e. we write v for v_i^t.

4.1.1 Basic Information Features

Inspired by the feature representation proposed in El Asri et al. (2016), we use a feature vector v_{basic} that is composed of binary sub-vectors to represent the basic information for each slot. Each slot has two value vectors: v_{value}^{sys} represents the value in the system state, and v_{value}^{user} represents the value in the user goal. Each value vector is a 4-dimensional one-hot vector, with coordinates encoding "none", "?", "don't care" or "other values", in this order. For example, in turn 1 in Fig. 2, for slot `Hotel-Price` $v_{value}^{user} = [1, 0, 0, 0]$, i.e., "none", because it is not in the user goal, and $v_{value}^{sys} = [0, 1, 0, 0]$, i.e., "?", because the system requests it.

The slot type vector v_{type} is a 2-dimensional vector which represents whether a slot is in the user goal as a constraint or a request. For example,

in Fig. 2 for Hotel-Area $v_{type} = [1, 0]$ (constraint), while for Hotel-Name $v_{type} = [0, 1]$ (request). A value of $[0, 0]$ means that the slot is not included in the user goal.

The state vector v_{ful} encodes whether or not a constraint or informable slot has been fulfilled. The value is set to 1 if the constraint has been fulfilled, and to 0 otherwise. The vector v_{first} similarly encodes whether a slot is mentioned for the first time.

The basic information feature vector v_{basic} is the concatenation of these vectors, i.e.,

$$v_{basic} = v^{user}_{value} \oplus v^{sys}_{value} \oplus v_{type} \oplus v_{ful} \oplus v_{first} \quad (2)$$

4.1.2 System Action Features

The system action feature vector v^{system}_{action} encodes system actions in each turn. There are two kinds of system actions, general actions and domain-specific actions. The general actions are composed only with general intents, such as "reqmore" and "bye". For example, general-reqmore(). The feature vector of general actions v_{gen} is a multi-hot encoding of whether or not a general intent appears in the dialogue. With a total number of n_{gen} general intents, for each $k \in \{1, \ldots, n_{gen}\}$, the k-th entry of v_{gen} is set to 1 if the k-th general intent is part of the system act.

On the other hand, domain-specific actions are composed with domains, slots, values, and domain-specific intents such as "recommend" and "select". For example, Recom(Hotel-Area=South). Each domain-specific action vector v_{spec_j} with the domain-specific j-th intent, $j \in \{1, \ldots, n_{spec}\}$, where n_{spec} is the total number of domain-specific intents, is represented by a 3-dimensional one-hot encoding that describes whether the value is "none", "?" or "other values".

The final action representation v^{system}_{action} is formed by concatenating n_{spec} domain-specific action representations together with the general action representation, i.e.,

$$v^{system}_{action} = v_{spec_0} \oplus \cdots \oplus v_{spec_{n_{spec}}} \oplus v_{gen}. \quad (3)$$

For the slot Hotel-Area in Fig. 3, we have a vector for each intent. For the intent "recommend" $v_{spec_0} = [0, 0, 1]$, which means that "other values" (in this case South) are mentioned. For all other domain-specific intents, the vectors are $[1, 0, 0]$ since no value is mentioned. In terms of the general intents, only "reqmore" is mentioned, so $v_{gen}[1] = 1$, as "reqmore" is the first general intent.

4.1.3 User Action Features

The output vector from the previous turn O^{t-1} is also included in the input features of the next turn t to take into account what has been mentioned by the US itself, i.e. for slot s_i in turn t, the user action feature $v^{user}_{action} = o^{t-1}_i$.

4.1.4 Domain and Slot Index Features

In some cases, multiple slots may share the same basic feature v_{basic}, system action feature v^{system}_{action} and user action feature v^{user}_{action}. This similarity in features of different slots makes it difficult for the model to distinguish one slot from another, despite the positional encoding. In particular, it is challenging for the model to learn the relationship between turns for a given slot because the number and the order of slots vary from one turn to the next. This may lead to over-generation: the model selects all slots with the same feature vector.

To counteract this issue, we introduce the index feature v_{index}, which consists of the domain index feature $v^{domain}_{index} \in \{0, 1\}^{l_d}$ and the slot index feature $v^{slot}_{index} \in \{0, 1\}^{l_s}$, where l_d is the maximum number of domains in a user goal and l_s is the maximum number of slots in any given domain[3].

To make the index feature ontology-independent, for a particular slot, v_{index} remains consistent throughout a dialogue, but varies between dialogues. The order of the index in each dialogue is determined by the order in the user goal. For example, the "hotel" domain can be the first domain in one user goal of the first dialogue, and the second domain in the next.

Then for each slot in each turn the input feature vector v is formed by concatenating all sub-vectors:

$$v = v_{basic} \oplus v^{system}_{action} \oplus v^{user}_{action} \oplus v_{index}. \quad (4)$$

An example of v for slot Hotel-Area is shown in Fig. 3 based on the dialogue history in Fig. 2. Examples of how the feature representation is constructed can be seen in Appendix D.

4.2 Domain Prediction

Inspired by solving downstream tasks using BERT (Devlin et al., 2019), we utilise the output of [CLS], o_{CLS}, to predict which domains are considered in turn t as a multi-label classification

[3]This does not need to be dependent on the number of domains or slots, it can simply be a random identifier assigned to each slot during one dialogue.

problem. The domain loss $loss_{domain}$ measures the difference between the output o_{CLS} and the target y_{CLS} for each turn by binary cross entropy (BCE). The final loss function is defined as

$$loss = loss_{slots} + loss_{domain}. \quad (5)$$

5 Experimental Setup

5.1 Supervised Training for TUS

Our model is implemented in PyTorch (Paszke et al., 2019) and optimised using the Adam optimiser (Kingma and Ba, 2015) with learning rate 5×10^{-4}. The dimension of the input linear layer is 100, the number of the transformer layers is 2, and the dimension of the output linear layer is 6. The maximum number of domains l_d is 6 and the maximum number of slots in one domain l_s is 10. During training, the dropout rate is 0.1.

We train our model[4] on the MultiWOZ 2.1 dataset (Eric et al., 2020), consisting of dialogues between two humans, one posing as a user and the other as an operator. The dialogues in the dataset are complex because there may be more than one domain involved in one dialogue, even in the same turn. During training and testing with the dataset, the order of slots in the input list is derived from the data, which means slot s_i is before slot s_{i+1} if the user mentioned slot s_i first. For inference without the dataset, such as when using TUS to train a dialogue policy, the order of slots is randomly generated.

We measure how well a US can fit the dataset by precision, recall, F1 score, and turn accuracy. The turn accuracy measures how many model predictions per turn are identical to the corpus, based on the oracle dialogue history.

5.2 Training Policies with USs

User simulators are designed to train dialogue systems, thus a better user simulator should result in a better dialogue system. We train different dialogue policies by proximal policy optimization (PPO) (Schulman et al., 2017), a simple and stable reinforcement learning algorithm, with ABUS, VHUS, and TUS as USs in the ConvLab-2 framework (Zhu et al., 2020). The policies are trained for 200 epochs, each of which consists of 1000 dialogues. The reward function gives a reward of 80 for a successful dialogue and of -1 for each dialogue turn, with the maximum number of dialogue turns set to 40. For failed dialogues, an additional penalty is set to -40. Each dialogue policy is trained on 5 random seeds. The dialogue policies are then evaluated using all USs by cross-model evaluation (Schatztnann et al., 2005) to demonstrate the generalisation ability of the policy trained with a particular US when evaluated with a different US.

5.3 Leave-one-domain-out Training

To evaluate the ability of TUS in handling unseen domains, we remove one domain during supervised learning of TUS. The leave-one-domain-out TUSs are used to train dialogue policies with all possible domains. For example, TUS-noHotel is trained on the dataset without the "hotel" domain. During policy training, the user goal is generated randomly from all possible domains.

Some domains in MultiWOZ may share the same slots, such as "restaurant" and "hotel" domains which contain property-related slots, e.g. "area," "name," and "price range." However, the corpus also includes domains that are quite different from the rest, For example, the "train" domain which contains many time-related slots such as "arrival time" or "departure time", as well as unique slots such as "price" and "duration." The different properties of the domains will allow us to study the zero-shot transfer capability of the model.

5.4 Human Evaluation

Following the setting in Kreyssig et al. (2018), we select 2 of the 5 trained versions of each dialogue policy for evaluation in a human trial: the version performing best on ABUS, and the version performing best in interaction with TUS. The results of the two versions are averaged. For each version we collect 200 dialogues, which means there are 400 dialogues for each policy in total. Dialogue policies trained with VHUS significantly underperform, so we only consider policies trained with ABUS or TUS for the human trial (see Table 1). The best and the worst policies in the leave-one-domain-out experiment are also included to see the upper and lower bound of the zero-shot domain generalisation performance.

Human evaluation is performed via DialCrowd (Lee et al., 2018) connected to Amazon Mechanical Turk[5]. Users are provided with a randomly generated user goal and are required to interact with our systems in natural language.

[4] https://gitlab.cs.uni-duesseldorf.de/general/dsml/tus_public

[5] https://www.mturk.com/

US for training	US for evaluation ABUS	VHUS	TUS	avg.
ABUS	0.93	0.09	0.58	0.53
VHUS	0.62	0.11	0.37	0.36
TUS	0.79	0.10	0.69	0.53

Table 1: The success rates of policies trained on ABUS, VHUS, and TUS when tested on various USs.

6 Experimental Results

6.1 Cross-model Evaluation

The results of our experiments are shown in Table 1. The policy trained with TUS performs well when evaluated with ABUS, with 10% absolute improvement in the success rate over its performance on TUS. On the other hand, while a policy trained with ABUS performs almost perfectly when evaluated with ABUS, the performance drops significantly, by 35% absolute, when this policy interacts with TUS. This signals that, in the case of ABUS, the policy overfits to the US used for training, and is not able to generalise well to the behaviour of other USs. We found that VHUS is neither able to train nor to evaluate a multi-domain policy adequately. This was also observed in the experiments by Takanobu et al. (2019). We suspect that this is due to the fact that VHUS was designed to operate on a single domain and does not generalise well to the multi-domain scenario. To the best of our knowledge, no other data-driven US has been developed for the multi-domain scenario.

The success rates of policies trained with ABUS and TUS during training, evaluated with both US, are shown in Fig. 4. Each of the systems is trained 5 times on different random seeds. We report the average success rate as well as the standard deviation. Although the policy trained with TUS is more unstable when evaluated on ABUS, it still shows an improvement from the initial policy, converging at around 79%. On the other hand, the policy trained with ABUS and evaluated with TUS barely show any improvements.

6.2 Impact of features and loss functions

We conduct an ablation study to investigate the usefulness of the proposed features and loss functions. The result is shown in Table. 2. First, we measure the performance of the basic model which uses only the basic information feature v_{basic}, the system action feature v_{action}^{system}, and the user action feature v_{action}^{user} as the input. While this model can

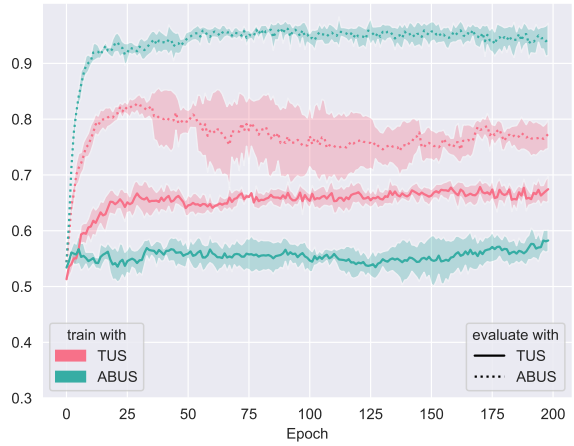

Figure 4: The success rates of policies during training with TUS and ABUS.

method	P	R	F1	ACC	LEN
basic model	0.11	0.71	0.19	0.11	4.51
+ index feature	0.17	0.51	0.26	0.44	1.29
+ domain loss	0.17	0.54	0.26	0.46	1.22

Table 2: The TUS ablation experiments. We analyse the impact of different settings by measuring precision P, recall R, F1 score, turn accuracy ACC, and the average slots mentioned in the first turn user action LEN. Humans, on average, mention 1.5 slots in the first turn.

have a high recall rate, the precision and the turn accuracy are fairly low. We deduce that without the index features the model cannot distinguish the difference between slots and therefore tends to select slots of the same slot type in one turn. For example, it provides all constraints in the first turn, which leads to high recall and over-generation.

Analysis of the generated user actions shows that the basic model tends to mention four or more slots in the first turn. This is unnatural, since human users tend to only mention one or two slots at the beginning of a dialogue. More details about the average slots per turn can be found in Appendix B.

After adding the index feature v_{index}, the recall rate is decreased by 17% absolute, but the turn accuracy is increased by 35% absolute, along with improvements on the precision and the F1 score. Furthermore, the average number of slots per turn is closer to that of a real user. Although the recall rate with respect to the target in the data is decreased, this is not necessarily a concern since in dialogue there are many different plausible actions for a given context. For example, when searching for a restaurant, we may provide the information of the area first, or the food type. The order of

US for training	removed data(%)	ABUS						TUS						mean
		Attr.	Hotel	Rest.	Taxi	Train	all	Attr.	Hotel	Rest.	Taxi	Train	all	
TUS-noAttr	32.20	**0.69**	0.64	**0.81**	0.65	**0.75**	**0.77**	0.71	0.58	0.66	0.61	**0.69**	0.69	**0.73**
TUS-noTaxi	19.60	0.63	0.61	0.81	0.61	0.70	0.74	0.69	0.60	**0.69**	0.64	0.68	**0.69**	0.72
TUS-noRest	45.21	0.62	**0.66**	0.80	0.56	0.75	0.76	**0.71**	**0.60**	0.64	**0.65**	0.64	0.68	0.72
TUS-noTrain	36.95	0.64	0.65	0.78	**0.67**	0.62	0.73	0.67	0.54	**0.63**	0.64	0.58	**0.64**	0.68
TUS-noHotel	40.15	0.59	0.59	0.76	**0.61**	0.54	0.69	0.64	0.52	0.61	0.61	0.55	0.62	0.66
TUS	0	0.69	0.68	0.81	0.66	0.77	0.79	0.73	0.59	0.66	0.68	0.64	0.69	0.74

Table 3: The success rates of dialogue policies trained with leave-one-domain-out TUSs. For example, the TUS-noAttr model is trained without the "attraction" domain. The sum of all removed data is more than 100% because some dialogues have multiple domains. We report results on all domains.

communicating these constraints may vary.

When we include the domain loss $loss_{domain}$ during training, both the recall rate and the turn accuracy improve while a similar average slot length per turn is maintained. These results indicate that the proposed ontology-independent index features can help the model to distinguish one slot from the other, which solves the over-generation problem of the basic model. The domain loss allows for more accurate prediction of the domain at turn level and the value for each slot at the same time.

6.3 Zero-shot Transfer

We test the capability of the model to handle unseen domains in a zero-shot experiment. In a leave-one-domain-out fashion we remove dialogues involving one particular domain when training the US. The share of each domain in the total dialogue data ranges from 19.60% to 45.21%. During dialogue policy training we sample the user goal from all domains. As presented in Table 3, removing one domain from the training data when training the US does not dramatically influence the policy on the corresponding domain. The final performance of the policies trained with leave-one-domain-out TUSs is still reasonably comparable to the policy trained with the full TUS. This is especially noteworthy considering the substantial amount of data removed during US training and the difference between each domain.

We observe that the model is able to learn about the removed domain from the other domains, although the removed domain is different from the remaining ones. For example, the "train" domain is very different from "attraction", "restaurant", and "hotel", and it is more complex than "taxi", but TUS-noTrain still performs reasonably well on the "train" domain. This signals that the model can do zero-shot transfer by leveraging other domain information. The worst performance on the "train" domain happens instead when the "hotel" domain is removed, i.e. the domain with the most substantial amount of data.

US for training	success			overall
	Attr.	Hotel	all	
ABUS	0.76	0.70	0.83	3.90
TUS	0.73	0.69	0.83	4.03
TUS-noAttr	0.75	0.54	0.81	4.01
TUS-noHotel	0.73	0.55	0.76	3.86

Table 4: The human evaluation results include success rate and overall rating as judged by users.

Our results also show that that some domains are more sensitive to data removal than others, irrespective of which domain is removed. This indicates that some domains are more involved and simply require more training data. This result demonstrates that TUS has the capability to handle new unseen domains without modifying the feature representation or retraining the model. It also shows that our model is sample-efficient.

6.4 Human Evaluation

The result of the human evaluation is shown in Table 4. In total, 156 users participated in the human evaluation. The number of interactions per user ranges from 10 to 80. The success rate measures whether the given goal is fulfilled by the system and the overall rating grades the system's performance from 1 star (poor) to 5 stars (excellent). TUS is able to achieve a comparable success rate as ABUS, without domain-specific information, and even scores slightly better in terms of overall rating. We were not able to observe any statistically significant differences between ABUS and TUS in the human evaluation. For leave-one-domain-out mod-

els, the performance of TUS-noAttr is similar to that one of ABUS and TUS without a statistically significant difference. We do however observe a statistically significant decrease in the success rate of TUS-noHotel when compared to TUS and ABUS ($p < 0.05$). This is unsurprising as the hotel domain accounts for 40.15% of the training data. For both TUS-noAttr and TUS-noHotel, the success rate on the domain "attraction" is comparable to TUS and ABUS, but the success rate on the domain "hotel" is relatively low. As observed in the simulation, removing a domain does not decrease the success rate in the corresponding domain as the feature representation is domain agnostic. Instead, it impacts domains which need plenty of data to learn.

7 Conclusion

We propose a domain-independent user simulator with transformers, TUS. We design ontology-independent input and output feature representations. TUS outperforms the data-driven VHUS and it has a comparable performance to the rule-based ABUS in cross-model evaluation. Human evaluation confirms that TUS can compete with ABUS even though ABUS is based on carefully designed domain-dependent rules. Our ablation study shows that the proposed features and loss functions are essential to model natural user behavior from data. Lastly, our zero-shot study shows that TUS can handle new domains without feature modification or model retraining, even with substantially fewer training samples.

In future work, we would like to learn the order of slots and add output language generation to make the behaviour of TUS more human-like. Applying reinforcement learning to this model would also be of interest.

Acknowledgments

We would like to thank Ting-Rui Chiang and Dr. Maxine Eskenazi from Carnegie Mellon University for their help with the human trial. This work is a part of DYMO project which has received funding from the European Research Council (ERC) provided under the Horizon 2020 research and innovation programme (Grant agreement No. STG2018 804636). N. Lubis, C. van Niekerk, M. Heck and S. Feng are funded by an Alexander von Humboldt Sofja Kovalevskaja Award endowed by the German Federal Ministry of Education and Research. Computational infrastructure and support were provided by the Centre for Information and Media Technology at Heinrich Heine University Düsseldorf. Computing resources were provided by Google Cloud.

References

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.

Heriberto Cuayáhuitl, Steve Renals, Oliver Lemon, and Hiroshi Shimodaira. 2005. Human-computer dialogue simulation using hidden markov models. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.*, pages 290–295. IEEE.

Lucie Daubigney, Matthieu Geist, Senthilkumar Chandramohan, and Olivier Pietquin. 2012. A comprehensive reinforcement learning framework for dialogue management optimization. *IEEE Journal of Selected Topics in Signal Processing*, 6(8):891–902.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Wieland Eckert, Esther Levin, and Roberto Pieraccini. 1997. User modeling for spoken dialogue system evaluation. In *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pages 80–87. IEEE.

Layla El Asri, Jing He, and Kaheer Suleman. 2016. A sequence-to-sequence model for user simulation in spoken dialogue systems. *Interspeech 2016*, pages 1151–1155.

Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, Adarsh Kumar, Anuj Goyal, Peter Ku, and Dilek Hakkani-Tur. 2020. Multiwoz 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 422–428.

Aciel Eshky, Ben Allison, and Mark Steedman. 2012. Generative goal-driven user simulation for dialog

management. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 71–81, Jeju Island, Korea. Association for Computational Linguistics.

Milica Gašić, Filip Jurčíček, Blaise Thomson, Kai Yu, and Steve Young. 2011. On-line policy optimisation of spoken dialogue systems via live interaction with human subjects. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 312–317. IEEE.

Kallirroi Georgila, James Henderson, and Oliver Lemon. 2006. User simulation for spoken dialogue systems: Learning and evaluation. In *Ninth International Conference on Spoken Language Processing*.

Izzeddin Gür, Dilek Hakkani-Tür, Gokhan Tür, and Pararth Shah. 2018. User modeling for task oriented dialogues. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 900–906. IEEE.

Simon Keizer, Milica Gašić, Filip Jurcicek, François Mairesse, Blaise Thomson, Kai Yu, and Steve Young. 2010. Parameter estimation for agenda-based user simulation. In *Proceedings of the SIGDIAL 2010 Conference*, pages 116–123.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Florian Kreyssig, Iñigo Casanueva, Paweł Budzianowski, and Milica Gašić. 2018. Neural user simulation for corpus-based policy optimisation of spoken dialogue systems. In *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, pages 60–69, Melbourne, Australia. Association for Computational Linguistics.

Kyusong Lee, Tiancheng Zhao, Alan W. Black, and Maxine Eskenazi. 2018. DialCrowd: A toolkit for easy dialog system assessment. In *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, pages 245–248, Melbourne, Australia. Association for Computational Linguistics.

Esther Levin and Roberto Pieraccini. 1997. A stochastic model of computer-human interaction for learning dialogue strategies. In *Fifth European Conference on Speech Communication and Technology*.

Xiujun Li, Zachary C Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. 2016. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Olivier Pietquin, Matthieu Geist, Senthilkumar Chandramohan, and Hervé Frezza-Buet. 2011. Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Transactions on Speech and Language Processing (TSLP)*, 7(3):1–21.

Jost Schatzmann, Kallirroi Georgila, and Steve Young. 2005. Quantitative evaluation of user simulation techniques for spoken dialogue systems. In *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*, pages 45–54.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152, Rochester, New York. Association for Computational Linguistics.

Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review*, 21(2):97–126.

Jost Schatztnann, Matthew N Stuttle, Karl Weilhammer, and Steve Young. 2005. Effects of the user model on simulation-based learning of dialogue strategies. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.*, pages 220–225. IEEE.

Konrad Scheffler and Steve Young. 2002. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *Proceedings of the second international conference on Human Language Technology Research*, pages 12–19. Citeseer.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Weiyan Shi, Kun Qian, Xuewei Wang, and Zhou Yu. 2019. How to build user simulators to train rl-based dialog systems. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1990–2000.

Ryuichi Takanobu, Hanlin Zhu, and Minlie Huang. 2019. Guided dialog policy learning: Reward estimation for multi-domain task-oriented dialog. In

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 100–110, Hong Kong, China. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Steve Young. 2002. Talking to machines (statistically speaking). In *Seventh International Conference on Spoken Language Processing*.

Qi Zhu, Zheng Zhang, Yan Fang, Xiang Li, Ryuichi Takanobu, Jinchao Li, Baolin Peng, Jianfeng Gao, Xiaoyan Zhu, and Minlie Huang. 2020. ConvLab-2: An Open-Source Toolkit for Building, Evaluating, and Diagnosing Dialogue Systems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

A All System Intents

All system intents in the MultiWOZ 2.1 dataset are listed in Table 5, including 5 general intents and 9 domain-specific intents.

type	intents
general	welcome, reqmore, bye, thank, greet
domain-specific	recommend, inform, request, select, book, nobook, offerbook, offerbooked, nooffer

Table 5: All system intents in the MultiWOZ 2.1

B Average Action Length in Each Turn

The average number of slots mentioned by TUS in each turn when interacting with the rule-based dialogue system is shown in Fig. 5. When the index feature v_{index} and the domain loss $loss_{domain}$ are added, TUS can deal with the over-generation problem and behave more similarly to what is observed in the corpus.

C Success Rates of Leave-one-domain-out Training

The training success rates of dialogue policies trained with leave-one-domain-out TUSs, which are evaluated on TUS, are shown in Fig. 6. In comparison to the full TUS, the leave-one-domain-out TUSs are more unstable, but they can achieve a comparable success rate at the end.

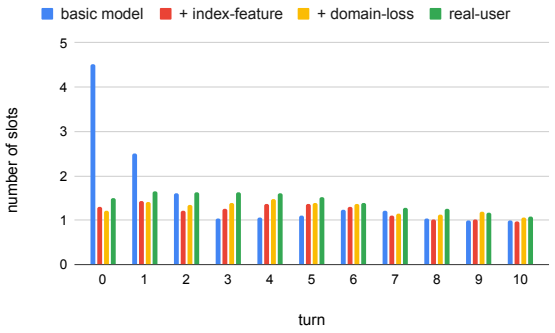

Figure 5: The average user action length per turn when interacting with the rule-based dialogue system. The average action length of real users in the corpus is also presented.

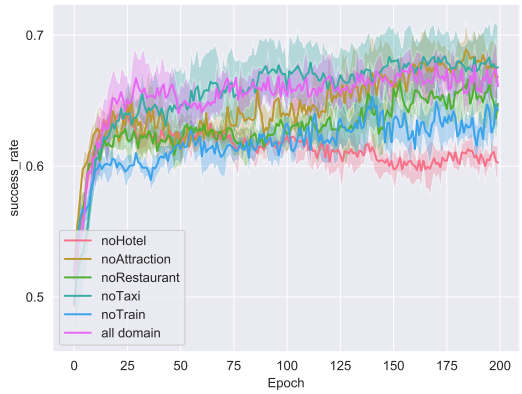

Figure 6: The success rates of dialogue policies trained with leave-one-domain-out TUSs during training, when evaluated on TUS.

D An example for the input feature representation

The list of input feature vectors and output sequence are presented on Fig. 7 based on Fig. 2.

For turn 0, V^0 only includes 4 vectors from the user goal. For turn 1, the system mentions slot `Hotel-Price`, which is not in the user goal, so the feature vector of slot `Hotel-Price` is inserted into V^1, where the 1-st dimension of v_{slot}^{domain} is 1 because domain *Hotel* is the first domain in this conversation and the 3-rd dimension of v_{index}^{slot} is 1 because it is the third slot in domain *Hotel*.

In comparison between the feature vectors of slot `Hotel-Area` in turn 0, v_1^0, and turn 1, v_1^0, the v_{value}^{sys} and v_{spec_0} are different because of the system's domain-specific action `Recom(Hotel-Area=South)`. The system also mentioned a general action, `general-reqmore()`, thus v_{gen} is changed. In

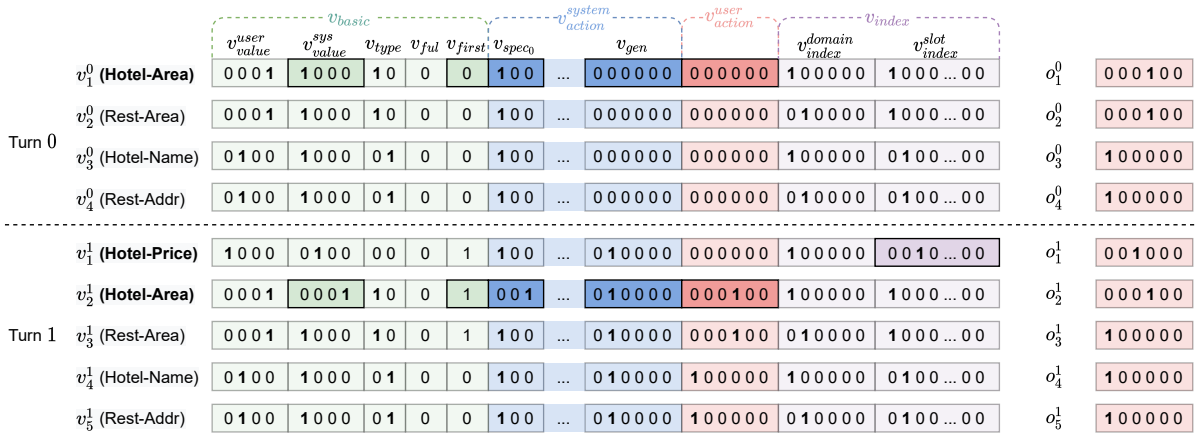

Figure 7: The input and feature representation according to Fig. 2. v_{CLS} and v_{sep} are ignored in this graph.

addition, this slot is first mentioned at turn 0, so v_{first} is changed from 0 to 1. Similarly, v^{user}_{action} is also modified according to the user action. On the other hand, v^{user}_{value} is the same because the user does not update its goal, v_{type} is not changed because the slot is still a constraint, and v_{ful} is 0 because it has not been fulfilled yet. v^{domain}_{index} and v^{slot}_{index} are also the same through the whole conversation.

E Example Dialogue Generated by TUS

An example dialogue with a multi-domain user goal is shown in Fig. 8. It shows that TUS is able to switch between different domains (from turn 2 to 6), respond to the system's requests, and generate multi-domain actions (in turn 5).

```
Turn 0
USR: Inform(Hotel-Area=north, Hotel-Stars=0,
            Hotel-Parking=yes)
SYS: Inform(Hotel-Parking=yes)

Turn 1
USR: Request(Hotel-Type)
SYS: Inform(Hotel-Type=guesthouse)
Turn 2
USR: Inform(Hotel-Stars=0, Hotel-Parking=yes)
SYS: Inform(Hotel-Stars=0, Hotel-Type=guesthouse,
            Hotel-Area=north, Hotel-Parking=yes,
            Hotel-Price=cheap)
Turn 3
USR: Inform(Attr-Type=college)
SYS: Inform(Attr-Choice=18),
     Recom(Attr-Name=hughes hall)
Turn 4
USR: Request(Taxi-Phone, Taxi-Car)
SYS: Request(Taxi-Leave)
Turn 5
USR: Inform(Taxi-Leave=dontcare),
     Request(Taxi-Phone, Taxi-Car, Attr-Fee)
SYS: Inform(Taxi-Car=audi, Taxi-Phone=44162528555,
            Taxi-Car=honda, Taxi-Phone=46793705737,
            Attr-Fee=free)
Turn 6
USR: Request(Attr-Post)
SYS: Inform(Attr-Post=cb23bu)
Turn 7
USR: general-bye()
SYS: general-greet()
```

Figure 8: A dialogue generated by TUS when interacting with the rule-based policy.