Lecture 13

Multimodal Virtual Assistants

By Giovanni Campagna, Jackie Yang, and Monica Lam
Goals of this Lecture

• Multimodal: Today’s software is GUI Based
  • How do we combine the best of GUI and Voice?
• RUSS: Rapid Universal Support System
  • Think out of the box: A “meta” agent that reads instructions
  • Technology: Learn how to use semantic parsing
• DoItHere
  • Think out of the box: Mobile devices are NOT small computers
  • Technology: Natural combination of voice and touch
Most Common Chatbots

• Customer service
  • The agents are often using the same web interface
  • They can do what consumers do
  • But people like to call – why?

• Call agent written with dialogue trees + intent classification
  • Trained for each task
Webtask Instructions in NL

Grounding Open-Domain Instructions to Automate Web Support Tasks
Nancy Xu, Sam Masling, Michael Du, Giovanni Campagna,
Larry Heck, James Landay, Monica S Lam
Proceedings of the NAACL-HLT, June 2021.

Customer Service Web Page

Help instructions

Help & Customer Service

Gifts, Gift Cards, and Registries › Gift Cards

Redeem a Gift Card

When you redeem an Amazon.com Gift Card or gift voucher to your account, the funds are stored in Your Account and will automatically apply to your next eligible order.

To redeem a gift card:
1. Find the claim code.
2. Go to Redeem a Gift Card.
3. Enter your claim code and select Apply to Your Balance.
Many Other Examples

| Create an Account |
|-------------------|
| 1. Select the **Account** button in the upper-right corner of Walmart.com |
| 2. Click **Create an Account** |
| 3. Enter your name and email address on the sign-up screen |
| 4. Create a password and confirm it |
| 5. Select **Create Account** |

| Redeem a Gift Card |
|--------------------|
| 1. Find the claim code. |
| 2. Go to **Redeem a Gift Card**. |
| 3. Enter your claim code and select **Apply to Your Balance**. |

| Find your AD Account |
|----------------------|
| 0. Click **Ads** to open Ads Manager |
| 0. Click **Reporting** |
| 0. Click the Ads tab under the graph |
| 0. Scroll down to the list of ads |
| 0. The 12-digit number beginning with 687 under the ad name is the ad ID |

Can we train one universal assistant that reads arbitrary instructions, interacts with users, and fills in the form?
Webtask Instructions in NL

Customer Service Web Page

Redeem a Gift Card

Help instructions

Agent instructions
1. Ask the user for the claim code
2. Go to Redeem a Gift Card.
3. Enter the claim code and select Apply to Your Balance.

Quiz: How?
RUSS (Rapid universal support system)

Natural Language instructions are parsed to an intermediate DSL (ThingTalk) that can be executed interactively in a DOM.
(1) Semantic Parsing: 6 Actions for Most Web Instructions

| Agent Action            | Description                                                                 |
|-------------------------|-----------------------------------------------------------------------------|
| @goto(url)              | Navigate to the given URL                                                   |
| @enter(element_id, dict_key) | Find the closest match to the given dictionary key and enter its value in the given input element |
| @click(element_id)      | Click on the given element                                                  |
| @read(element_id)       | Read the content of the given element to the user                          |
| @say(message)           | Read the given message to the user                                          |
| @ask(dict_key)          | Ask the user for the value of a dictionary key                              |

*element_id*: element ID in the DOM (Document Object Model)
Hardest Part: From NL description for human to operation using \text{ElementIds}

Given a web page:

\textbf{DOM}:

\begin{itemize}
  \item element\_id: 1, type = "body"
  \item element\_id: 2, type = "h1", text = "Your Orders"
  \item element\_id: 3, type = "form"
  \item \ldots
  \item element\_id: 48, type = "label", text = "order number"
  \item element\_id: 49, type = "input"
\end{itemize}

\textbf{From}:

\textbf{Instruction}: “Enter the user’s order number in the text field that says order number”

\textbf{To}:

\textbf{Action}: \@enter(\text{text = order\_number, element = 49})
Two Steps: Add Intermediate Function: Retrieve

**DOM:**
- `element_id: 1, type = "body"
- `element_id: 2, type = "h1", text = "Your Orders"
- `element_id: 3, type = "form"
  - ...  
  - `element_id: 48, type = "label", text = "order number"
  - `element_id: 49, type = "input"
  - ...  

**Semantic Parser (1):**
- **Instruction:** “Enter the user’s order number in the text field that says order number”

**Grounding Model (2):**
- **Action:** `@enter(text = order_number, element = id)`

**ThingTalk:**
- `@retrieve(description = “order number”, type = input) 
  \Rightarrow @enter(text = order_number, element = id)`

**Grounding Function | Description**
|----------------------|--------------------------------------------------|
| `@retrieve(descr, type, loc, above, below, right_of, left_of) : element_id` | Retrieves the elements matching the descriptors, returns an `element_id`. |

formal description independent of the DOM
Design of Retrieve

Another Example

- Enter the user’s email (clause 3)
- in the text field under (clause 2)
- email address or username (clause 1)

Design Rationale

- Compositional
- Formal representation of NL
  - Independent of the DOM

Quiz: What is the benefit?
Grounding Model: ThingTalk -> DOM Element

@retrieve(
    description="order number",
    type=input
)

Features

  type=input

  description="order number"

DOM:

  element_id: 1, type = "body"
  element_id: 2, type = "h1", text = "Your Orders"
  element_id: 3, type = "form"

  ...

  element_id: 48, type = "label", text = "order number"
  element_id: 49, type = "input"

  ...

filter by type and location

encode in SentenceBERT and find max cosine similarities of embedding with embedding of input text description

element id = 49
Semantic Parser: Language -> ThingTalk

{“@click”: 0.03, “@enter”: 0.08, “description”: 0.06, “color”: 0.01, …}
Training

• Template-based synthesis method to generate sample instruction -> ThingTalk pairs.

• Total: 1.5M training samples from roughly 840 distinct templates.
Evaluation: The RUSS Evaluation Dataset

- 80 tasks
- 741 instructions
- 22 help centers
Entity Extraction

• Entity Strings → Placeholder tokens for parsing
• Example:
  Pre-parsing
  https://www.amazon.com/gp/help/customer/display.html?nodeId=G5D4TA7NBKQT7GW2 → “URL”

  Post-parsing
  “URL” → https://www.amazon.com/gp/help/customer/display.html?nodeId=G5D4TA7NBKQT7GW2
# Accuracy on Semantic Parsing

| Model                                      | Accuracy (test) |
|--------------------------------------------|-----------------|
| RUSS (1.5M training parses)                | 87.0%           |

| Ablations                                      | Accuracy (dev) |
|-----------------------------------------------|----------------|
| RUSS (1.5M training parses)                   | 88.2%           |
| - entity extraction                           | 77.6%           |
| - 1M training parses, entity extraction       | 70.0%           |
## Accuracy on Grounded Instructions

| Model                        | Grounding Acc (test) |
|------------------------------|----------------------|
| RUSS                         | 63.6%                |
| End-to-End Baseline          | 51.1%                |
| PhraseNode                   | 46.5%                |
End-To-End Evaluation: User Study

# 1 Redeem Amazon Gift Card
# 2 Get Pinterest Ad Account Number
# 3 Log out of all Spotify accounts
# 4 Create new Walmart account
# 5 Send Google feedback
Summary of Russ

• A meta-agent
  • Trained once, run on any web service instructions
  • Read instructions → answer calls immediately

• Design of ThingTalk
  • Separate problem into semantic parsing + grounding model

• Performance
  • 76.6% end-to-end accuracy
  • 69% of users prefer RUSS over following web instructions

• 1 quarter project!
DoThisHere

Multimodal Interaction to Improve Cross-Application Tasks on Mobile Devices

Jackie (Junrui) Yang
Monica S. Lam
James A. Landay
Users do many cross-app tasks on their smartphones

- 31.8% of smartphone usage sessions involved two or more apps [1]
- Our survey on MTurk (N=75) shows eight common categories of cross-app tasks

[1] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. 2011. Falling asleep with Angry Birds, Facebook and Kindle.
Users do many cross-app tasks on their smartphones
Current interfaces offers limited support for information transfer

For **simple information**, users have to read info and remember the result.

For **complex information**, users have to go back and forth between apps.

Hard to get it right with voice.
DoThisHere introduces: Query, Do, and Keep

For simple information, user can use Query and Do to transfer information

For complex information, user can keep part of the interface for reference
DoThisHere introduces: Query, Do, and Keep

For **simple information**, user can use Query and Do to transfer information.

For **complex information**, user can keep part of the interface for reference.

| Query | Do | Keep |
|-------|----|------|
| Insert my duo code here. | Show me the review on IMDB | Keep this. |

For complex information, user can keep part of the interface for reference.
DoThisHere can be portable to other platforms

DoThisHere is based on existing mobile infrastructure
DoThisHere can be portable to other platforms

DoThisHere is based on existing mobile infrastructure

Mitigates the “cold start problem”
DoThisHere can support majority of cross-apps tasks

- Among the cross-app tasks we collected, DoThisHere can help on 95% of the unsupported tasks.

![Chart showing support percentages for cross-app tasks.](chart.png)
DoThisHere may help reduce users’ cognitive load

- Among the cross-app tasks we collected, DoThisHere can help on 95% of the unsupported tasks.
- Our user study (N=12) indicated that DoThisHere may help on reducing user’s cognitive load (NASA-TLX).

* Task:
  1: Keep
  2, 3: Query
  4: Do

*: p< 0.05, **: p < 0.01
How can we better support complex data transfer?

How can we support more than what's currently supported in voice?
ReactGenie: toolkit for multimodal interaction apps

Jackie (Junrui) Yang, Hung Bui, Karina Li, Shuning Zhang, Monica Lam, James Landay
The benefit of multimodal interaction is beyond cross-app tasks

- Navigation
- Integration
- Selection

Share this video
Share this title
Share this account
ReactGenie tries to solve two problems of DoThisHere

Lack of skills

10+ actions in the app vs. 2 actions via voice

Lack of complex types on screen

More than just text!
ReactGenie is based on ReactNative

State
Restaurant
  OrderFoods
  Reserve
Cuisine
Address
Food
  Name
  Type
  Price
Account
  Email
  Name
  ProfilePicture

UI
Login Page
Restaurant Page
Food Detail Page
Shopping Cart
Account Page
ReactGenie is based on ReactNative

**Annotation + GenieLamp**

| State |
|-------|
| Restaurant |
| OrderFoods |
| Reserve |
| Cuisine |
| Address |

| Food |
|------|
| Name |
| Type |
| Price |

| Account |
|---------|
| Email |
| Name |
| ProfilePicture |

**Annotation + Mapping**

| UI |
|----|
| Login Page |
| Restaurant Page |
| Food Detail Page |
| Shopping Cart |
| Account Page |

Automatically generates skills

Automatically maps data and UI
GenieLamp (Genie Large Language Model Parser)

```java
@GenieObject
export class Restaurant extends LampObject {
  static _ol: Restaurant[] = undefined
  public orders: Order[] = [];
  static oll(): Restaurant[] {...}
}

@GenieFunction
descriptionO: {} {
  return {
    name: this.name,
    cuisine: this.cuisine,
    rating: this.rating,
    priceGrade: this.priceGrade,
    address: this.address
  }
}

@GenieFunction
createFood(name: string, price: number) {
  const food = new Food(name, price, this)
  this.menu.push(food);
  return food
}
```

```java
// Here are all the function that we have

class Restaurant {
  string name;
  string address;
  ...

  // Examples:
  user: get me the best restaurant in Palo Alto
  parsed: Restaurant.all().matching(field: .address, value: "Palo Alto")
  ...

  // Current User
  user: order the same burger that I ordered at McDonald last time
  parsed: Order.current.addFoods(foods: Order.all().matching(field:
```
Future work

• Working on from UI annotations to mapping between data and UI
• Building demo apps
  • Food ordering app
  • Messaging app
• Chart/Visualization library