Dialog Management
Dialog System & Architectures
Dialogue System

A system to provide *interface* between the user and a computer-based application

Interact on turn-by-turn basis

Dialogue manager

- Control the flow of the dialogue
- Main flow
  - information gathering from user
  - communicating with external application
  - communicating information back to the user

- Three types of dialogue system
  - finite state- (or graph-) based
  - *frame-based*
  - *agent-based*
Dialog System Architecture

• Typical dialog system has following components
  ► User Interface
    ◦ Input: Speech Recognition, keyboard, Pen-gesture recognition ..
    ◦ Output: Display, Sound, Vibration ..
  ► Context Interpretation
    ◦ Natural language understanding (NLU)
    ◦ Reference resolution
    ◦ Anaphora resolution
  ► Dialog Management
    ◦ History management
    ◦ Discourse management

• Many dialog system architectures are introduced.
  ► DARPA Communicator
  ► GALAXY Communicator
  ► etc.
The DARPA Communicator program was designed to support the creation of *speech-enabled interfaces* that scale gracefully across modalities, from speech-only to interfaces that include graphics, maps, pointing and gesture.
Galaxy Communicator

• The architecture

Diagram of Galaxy Communicator architecture:
- Language Generation
- Text-to-Speech Conversion
- Dialogue Management
- Application Backend
- Context Tracking
- Audio Server
- Speech Recognition
- Frame Construction

Diagram center is labeled Hub.
Galaxy Communicator

- Message Passing Protocol

```c
{ c FromAudio ... }

PROGRAM: FromAudio
RULE: ---> Recognizer.Recognize ...

Hub program

Audio
audio.mitre.org

{ c Recognizer.Recognize ... }

Speech Recognition
rec.mitre.org

Hub

Gal_Frame Recognize(Gal_Frame *f, void *server_data)
{
    ...
}

Dispatch function
```
Dialog System Approaches
There are many approaches to represent dialog:

- Frame based
- Agent based
- Voice-XML based
- Information State approach
Frame-based Approach

- Frame-based system
  - Asks the user questions to *fill slots in a template* in order to perform a task (form-filling task)
  - Permits the user to respond more flexibly to the system’s prompts (as in Example 2.)
  - Recognizes the main concepts in the user’s utterance

**Example 1)**
- **System:** What is your destination?
- **User:** London.
- **System:** What day do you want to travel?
- **User:** Friday

**Example 2)**
- **System:** What is your destination?
- **User:** London on Friday around 10 in the morning.
- **System:** I have the following connection …
Agent-based Approach

- Properties
  - Complex communication using unrestricted natural language
  - Mixed-Initiative
  - Co-operative problem solving
  - Theorem proving, planning, distributed architectures
  - Conversational agents

- Examples

  **User**: I’m looking for a job in the Calais area. Are there any servers?

  **System**: No, there aren’t any employment servers for Calais. However, there is an employment server for Pasde-Calais and an employment server for Lille. Are you interested in one of these?

  - System attempts to provide a more co-operative response that might address the user’s needs.
The TRIPS System Architecture
VoiceXML-based System

• What is VoiceXML?
  ► The HTML/XML of the voice web.
  ► *The open standard markup language for voice application*

• Can do
  ► Rapid implementation and management
  ► Integrated with World Wide Web
  ► Mixed-Initiative dialogue
  ► Able to input Push Button on Telephone
  ► Simple Dialogue implementation solution
Example - <Menu>

**Browser**: Say one of:

Sports scores; Weather information; Log in.

**User**: Sports scores

```xml
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <menu>
    <prompt>Say one of: <enumerate/></prompt>
    <choice next="http://www.example.com/sports.vxml">
      Sports scores
    </choice>
    <choice next="http://www.example.com/weather.vxml">
      Weather information
    </choice>
    <choice next="#login">
      Log in
    </choice>
  </menu>
</vxml>
```
Information State Approach

- A method of specifying a dialogue theory that makes it straightforward to implement
- Consisting of following five constituents
  - Information Components
    - Including aspects of common context
    - (e.g., participants, common ground, linguistic and intentional structure, obligations and commitments, beliefs, intentions, user models, etc.)
  - Formal Representations
    - How to model the information components
    - (e.g., as lists, sets, typed feature structures, records, etc.)

- Private: \[ \text{BEL} : \text{SET(Prop)} \]
- Agenda: \[ \text{AGENDA} : \text{STACK(Action)} \]
- Shared: \[ \begin{aligned}
&\text{BEL} : \text{SET(Prop)} \\
&\text{QUD} : \text{STACK(QUESTION)} \\
&\text{LM} : \text{MOVE}
\end{aligned} \]
Information State Approach

► Dialogue Moves
  ◦ Trigger the update of the information state
  ◦ Be correlated with externally performed actions

► Update Rules
  ◦ Govern the updating of the information state

- **U-RULE**: `integrateSysAsk`
  
  **PRE**: 
  \[
  \begin{aligned}
  &\text{val(\texttt{SHARED.LM, ask(usr, Q)})} \\
  &\text{fst(\texttt{PRIVATE.AGENDA, raise(Q)})}
  \end{aligned}
  \]

  **EFF**: 
  \[
  \begin{aligned}
  &\text{push(\texttt{SHARED.QUD, Q})} \\
  &\text{pop(\texttt{PRIVATE.AGENDA})}
  \end{aligned}
  \]

- **U-RULE**: `integrateUserAnswer`
  
  **PRE**: 
  \[
  \begin{aligned}
  &\text{val(\texttt{SHARED.LM, answer(usr, A)})} \\
  &\text{fst(\texttt{SHARED.QUD, Q})}
  \end{aligned}
  \]

  **DOM**: relevant(\(A, Q\))

  **DOM**: reduce(\(Q, A, P\))

  **EFF**: 
  \[
  \begin{aligned}
  &\text{add(\texttt{SHARED.BEL, P})}
  \end{aligned}
  \]

- **U-RULE**: `downdateQUD`
  
  **PRE**: 
  \[
  \begin{aligned}
  &\text{fst(\texttt{SHARED.QUD, Q})} \\
  &\text{in(\texttt{SHARED.BEL, P})}
  \end{aligned}
  \]

  **DOM**: resolves(\(P, Q\))

  **EFF**: 
  \[
  \begin{aligned}
  &\text{pop(\texttt{SHARED.QUD})}
  \end{aligned}
  \]

- **U-RULE**: `selectAsk`
  
  **PRE**: 
  \[
  \begin{aligned}
  &\text{fst(\texttt{PRIVATE.AGENDA, raise(Q)})}
  \end{aligned}
  \]

  **EFF**: 
  \[
  \begin{aligned}
  &\text{set(\texttt{NEXT\_MOVE, ask(Q)})}
  \end{aligned}
  \]

► Update Strategy
  ◦ For deciding which rules to apply at a given point from the set of applicable ones
Example Dialogue

PRIVATE =

AGENDA = \{raise(\?x.\text{dest-city}(x)), \text{raise}(\ldots), \ldots\}

SHARED =

BEL = \{

QUD = \langle \rangle

LM = \ldots

\}

\text{set}(\text{NEXT-MOVE}, \text{ask}(\?x.\text{dest-city}(x)))

\text{U-RULE: selectAsk}

\text{PRE:}\ \{\ \text{fst}(\text{PRIVATE.AGENDA}, \text{raise}(Q))\}

\text{EFF:}\ \{\ \text{set}(\text{NEXT-MOVE}, \text{ask}(Q))\}
Example Dialogue

Sys: Where do you want to go?

\[
\begin{align*}
\text{PRIVATE} &= \begin{bmatrix}
\text{BEL} &= \{\} \\
\text{AGENDA} &= \langle \text{raise(?x.dest-city(x)), raise(...), ...} \rangle \\
\text{BEL} &= \{\} \\
\text{QUD} &= \langle \rangle \\
\text{LM} &= \text{ask(?x.dest-city(x))}
\end{bmatrix} \\
\text{SHARED} &= \begin{bmatrix}
\text{push}(&\text{SHARED.QUD, ?x.dest-city(x)}) \\
\text{pop}(&\text{PRIVATE.AGENDA})
\end{bmatrix}
\end{align*}
\]

\text{U-RULE: integrateSysAsk}

\text{PRE:}
\begin{align*}
\text{val}(&\text{SHARED.LM, ask(usr,Q)}) \\
\text{fst}(\text{PRIVATE.AGENDA}, \text{raise}(Q))
\end{align*}

\text{EFF:}
\begin{align*}
\text{push}(&\text{SHARED.QUD, Q}) \\
\text{pop}(&\text{PRIVATE.AGENDA})
\end{align*}
Example Dialogue

\[
\begin{align*}
\text{PRIVATE} &= \begin{cases}
\text{BEL} & = & \{\} \\
\text{AGENDA} & = & \langle \text{raise(?x.depart} - \text{city}(x)), \ldots \rangle \\
\text{BEL} & = & \{\} \\
\text{QUD} & = & \langle \ ?x.\text{dest-city}(x) \rangle \\
\text{LM} & = & \text{ask(?x.\text{dest-city}(x))}
\end{cases} \\
\text{SHARED} &= \end{align*}
\]
Example Dialogue

Usr: Malvern

\[
\begin{align*}
\text{PRIVATE} &= \begin{cases} 
\text{BEL} = & \{ \} \\
\text{AGENDA} = & \{ \text{raise(?x.depart - city(x)), } \ldots \} 
\end{cases} \\
\text{SHARED} &= \begin{cases} 
\text{BEL} = & \{ \} \\
\text{QUD} = & \{ ?x.\text{dest-city}(x) \} \\
\text{LM} = & \text{answer(malvern)} 
\end{cases}
\end{align*}
\]

add(SHARED.BEL, dest-city(malvern))

U-RULE: integrateUserAnswer

\[
\begin{align*}
\text{val(SHARED.LM, answer(usr,A))}, \\
\text{fst(SHARED.QUD, Q)}
\end{align*}
\]

PRE: \{ 
- \text{DOMAIN :: relevant(A, Q)} \\
- \text{DOMAIN :: reduce(Q, A, P)}
\}

EFF: \{ 
- \text{add(SHARED.BEL, P)}
\}
Example Dialogue

PRIVATE =
AGENDA = \{\text{\textit{raise(?x.depart - city(x)), ...}}\}
BEL = \{\text{\textit{dest-city(malvern)}}\}
QUD = \{\text{\textit{?x.dest-city(x)}}\}
LM = \text{answer(malvern)}

\text{pop(SHARED.QUD)}

\text{U-RULE: downdateQUD}
\text{PRE:}\ 
\begin{align*}
\text{fst}(\text{SHARED.QUD}, Q) \\
\text{in}(\text{SHARED.BEL}, P) \\
\text{DOMAIN :: resolves}(P, Q)
\end{align*}
\text{EFF:}\ 
\begin{align*}
\text{pop}(\text{SHARED.QUD})
\end{align*}
Example Dialogue

\[
\begin{align*}
\text{PRIVATE} & = \begin{cases}
\text{BEL} & = \emptyset \\
\text{AGENDA} & = \langle \text{raise(?x.depart-city(x)), \ldots} \rangle \\
\text{BEL} & = \{\text{dest-city(malvern)}\} \\
\text{QUD} & = \langle \rangle \\
\text{LM} & = \text{answer(malvern)}
\end{cases} \\
\text{SHARED} & = \end{align*}
\]
Dialog Modeling Techniques
Reinforcement Learning

Training Info = desired (target) outputs

Objective: To minimize error (Target Output – Actual Output)

Inputs: (Feature, Target Label)

Supervised Learning System

Outputs

Training Info = evaluations (“rewards”/”costs”)

Objective: To get as much reward as possible

Inputs: (State, Action, Reward)

RL System

Outputs (“actions”)
Stochastic Modeling Approach

- Stochastic Dialog Modeling [E. Levin et al, 2000]
  - Optimization Problem
    - Minimization of Expected Cost ($C_D$)

\[ C_D = \sum_i \omega_i C_i \]

$C_i$ measures the effectiveness and the achievement of application goal

- Mathematical Formalization
  - Markov Decision Process
    - Defining State Spaces, Action Sets, and Cost Function
    - Formalize dialog design criteria as objective function

- Automatic Dialog Strategy Learning from Data
  - Reinforcement Learning
Mathematical Formalization

- Markov Decision Process (MDP)
  - Problems with *cost (or reward) objective function* are well modeled as *Markov Decision Process*.
  - The specification of a sequential decision problem for a fully observable environment that satisfies the Markov Assumption and yields additive rewards.

Dialog Management

1. Dialog Manager
2. Dialog State
3. Cost (Turn, Error, DB Access, etc.)
4. Environment (User, External DB or other Servers)
5. Dialog Action (Prompts, Queries, etc.)
Dialog as a Markov Decision Process

User goal $S_u$ → $a_u$ → Speech Understanding

→ Speech Generation $\tilde{a}_m$

→ State Estimator $\tilde{a}_u$ → $S_d$

→ Dialog Policy $\pi$

→ Reinforcement Learning $R = \sum_k \gamma^k r_k$

Optimize

Noisy estimate of user dialog act $\tilde{a}_u$

Machine state $\tilde{s}_m$

Machine dialog act $\tilde{a}_m$

[S. Young, 2006]
Month and Day Example

• State Space
  - State $S_t$ represents all the knowledge of the system at time $t$ (values of the relevant variables).
    - $S_t = (d, m)$ where $d=\ldots,-1,\ldots,31$ and $m=\ldots,-1,\ldots,12$
    - 0 : not yet filled
    - -1 : completely filled
    - (0,0) = Initial State
    - (-1,-1) = Final State
Month and Day Example

- State Space

1 (initial) + 12(months) + 31(days) + 365(dates) + 1(final)

Total Dialog State: 410 states
Month and Day Example

• Action Set
  - At each state, the system can choose an action $a_t$.
    ◦ Dialog Actions
      - Asking the user for input, providing a user some output, confirmation, etc.

Which month? ($A_m$)
Which day? ($A_d$)
Which date? ($A_{dm}$)
Thank you. Good Bye. ($A_f$)
Month and Day Example

• State Transitions
  ▶ When an action is taken the system changes its state.

SYSTEM : Which month?

- Month: 1
- Month: 11
Month: 12

New state might depend on external inputs:
Not Deterministic

Transition Probability: $P_{T}(S_{t+1}|S_{t},a_{t})$
Month and Day Example

- Action Costs and Objective Function
  - A cost $C_t$ is associated to action $a_t$ at state $S_t$.

SYSTEM : Which month?

Cost Distribution: $P_c(C_t|S_t,a_t)$

$$C_D = \omega_i \# \text{interactions} + \omega_e \# \text{Errors} + \omega_f \# \text{unfilled slots}$$
Month and Day Example

Strategy 1.

Good Bye.

\[ C_1 = \omega_i \times 1 + \omega_f \times 2 \]

Strategy 2.

Which date?

Day

Month

Good Bye.

\[ C_2 = \omega_i \times 2 + \omega_e \times 2 \times P_1 + \omega_f \times 0 \]

Strategy 3.

Which day?

Day

Which month?

Day

Month

Good Bye.

\[ C_3 = \omega_i \times 3 + \omega_e \times 2 \times P_2 + \omega_f \times 0 \]

Optimal strategy is the one that minimizes the cost.

Strategy 1 is optimal if \( w_i + P_2 \times w_e - w_f > 0 \) → Recognition error rate is too high

Strategy 3 is optimal if \( 2 \times (P_1 - P_2) \times w_e - w_i > 0 \)

→ \( P_1 \) is much more high than \( P_2 \) against a cost of longer interaction
Policy

• The goal of MDP is to learn a policy, \( \pi : S \rightarrow A \)
  
  - But we have no training examples of form \(<s,a>\)
  - Training examples are of form \(<s,a,s',r>\)
  - For selecting the next action \(a_t\) based on the current observed state \(s_t\).

Goal: Learn to choose actions that maximize the reward function.

\[
r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad (\text{where } 0 \leq \gamma < 1)
\]

discount factor
Policy

• Discounted Cumulative Reward
  ► Infinite-Horizon Model

\[ V^\pi(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad \text{(where } 0 \leq \gamma < 1) \]
  ◦ \( \gamma = 0 : V^\pi(s_t) = r_t \)
    – Only immediate reward considered.
  ◦ \( \gamma \) closer to 1 : Delayed Reward
    – Future rewards are given greater emphasis relative to the immediate reward.

• Optimal Policy (\( \pi^* \))
  ► Optimized policy \( \pi \) that maximize \( V^\pi(s) \) for all state \( s \).

\[ \pi^* \equiv \arg\max_\pi V^\pi(s) \quad \text{for } \forall s \]
Q-Learning

• Define the Q-Function.
  ► As evaluation function.

\[ Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a)) \]

• Rewrite the optimal policy.

\[
\pi^* = \arg\max_a \left( r(s, a) + \gamma V^*(\delta(s, a)) \right)
\]

\[
\pi^*(s) = \arg\max_a Q(s, a)
\]

• Why is this rewrite important?
  ► It shows that if the agent learns the Q-function instead of the \( V^* \) function.
    ◦ It will be able to select optimal actions even when it has no knowledge of the function \( r \) and \( \delta \).
Q-Learning

• How can Q be learned?
  ► Learning the Q function corresponds to learning the optimal policy.
    ◦ The close relationship between Q and V*
      
      \[ V^*(s) = \max_{a'} Q(s, a') \]
  ► It can be written recursively as
    
    \[ Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \]
    ◦ This recursive definition of Q provides the basis for algorithm that iteratively approximate Q.
  ► It can updates the table entry for Q(s,a) following each such transition, according to the rule.
    
    \[ \hat{Q}(s, a) \rightarrow r + \gamma \max_{a'} \hat{Q}(s', a') \]
Q-Learning

• Q-Learning algorithm for deterministic MDP.

For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state $s$

Do forever:
  • Select an action $a$ and execute it
  • Receive immediate reward $r$
  • Observe the new state $s'$
  • Update the table entry for $\hat{Q}(s, a)$ as follows:
    \[ \hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a') \]
  • $s \leftarrow s'$
Action Selection in Q-Learning

- How actions are chosen by the agent.
  - To select the action that maximize the Q hat function.
    - Thereby exploiting its current approximation Q hat.
    - *Biased* to previously trained Q hat function.

- Probability Assigning
  - Actions with higher Q hat values are assigned higher probabilities.
  - But every action is assigned a nonzero probability.

\[
P(a_i \mid s) = \frac{k \hat{Q}(s, a_i)}{\sum_j k \hat{Q}(s, a_j)}
\]

- \(k > 0\) is a constant that determines how strongly the selection favors actions with high Q hat values.
  - *Larger values of \(k\)* will assign higher probabilities to actions with above average Q hat.
    - Causing the agent to *exploit* what it has learned and seek actions it believes will maximize its reward.
  - \(k\) is varied with the number of iterations.
    - Exploitation vs. Exploration