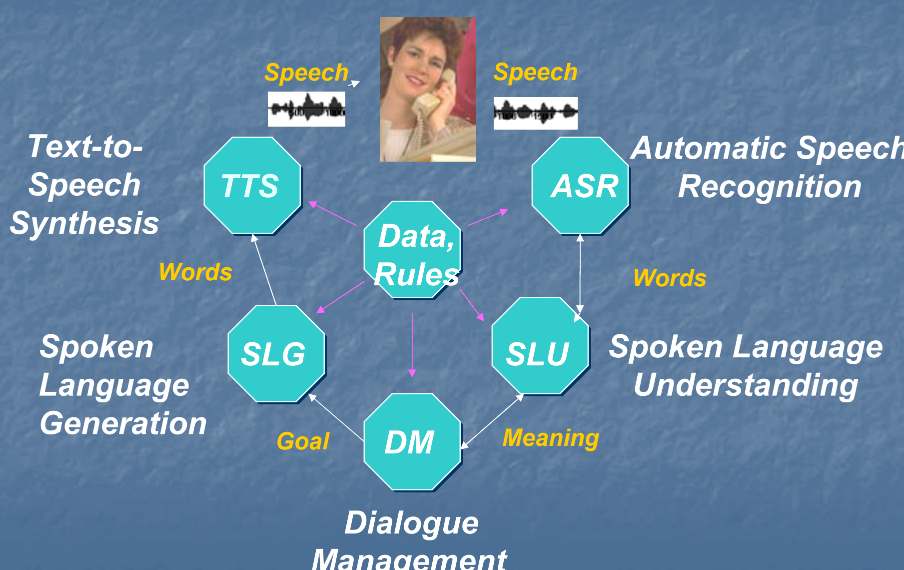
Architectures for Spoken Dialogue Systems/ Dialogue Management

Spoken Dialogue Systems



Which has the hardest job? Why?

- ASR – recognize the words the user spoke
- NLP – recognize the meaning of the user's utterance
- DM – decide what to answer
- NLG – formulate the answer in natural languuage
- TTS – speak the answer clearly

VXML: Strengths

- Simple, straightforward format
- Modelled on HTML, known technology
- Makes design/deployment of simple dialogue systems possible for developers with little/no background in ASR/NL
- Audio server can be hosted remotely, taking away further complication

VXML: Weaknesses

- State-based dialogue systems inherently limiting
- Underlying technologies typically also limited
  - Simple grammars—isolated words/phrases
  - Grammar-based ASR
  - Limited capabilities for language generation
  - Limited logic/reasoning for dialogue

What else might a spoken dialogue system need to account for?

Input from the Audio Server

- If barge-in is enabled, how is truncated input interpreted:

User: I'm interested in Thai restaurants in North London.
System: *I know of 8 Thai rest-*
User: Wait, that's not what I wanted.

User: I'm interested in Thai restaurants in North London.
System: *I know of 8 Thai restaurants in North London. There's Banh Mi, Thai Palace, Gold-*
User: Wait, that's the one I wanted.

Input from ASR

- Can dialogue state constrain recognition choice?

User: I'm going to Dallas on May eighteenth.
System: *Okay, where are you leaving from?*
User: Dulles.

User: I want to return on May twentieth.
System hears:
i want to return on may twelfth
i want to return on may twentieth
System: *So that's returning on May twelfth.*

What information does NLP use?

- Words/phrases are interpreted *in context*

User: I need to book a flight.
System: *Okay, where are you leaving from?*
User: Dulles.

How about NLG?

- Tailor response to fit user model/current history

User: I'm interested in Thai restaurants in North London.
System: *I know of 8 Thai restaurants in North London. Two of them have very high food quality: Banh Mi and Golden Siam.*

Can TTS use dialogue information?

- Emphasize new/pertinent information

User: I'm interested in Thai restaurants in North London.
System: *I know of 8 Thai restaurants in North London. Two of them have very high food quality: Banh Mi and Golden Siam.*
User: Actually, what about Chinese restaurants.
System: Okay, Chinese restaurants in North London.

What else goes into SDS?

- Meta-level responses
  - Dynamically generated help messages based on current state of dialogue/input/backend data
  - Summary descriptions of backend data
- Fallback mechanisms
  - Descriptive responses when user query results in NULL output from database
- Complex reasoning about domain/database
  - Intelligent ordering of database tuples
  - Incorporation of user preferences
  - Analysis of backend data in light of dialogue context

What else goes into SDS?

- Global data stores for reprocessing of system output across multiple turns
- Multimodal capabilities (ongoing work)
- Multilingual capabilities
- Learning

Morale: there's a lot to think about

- Dialogue systems involve individually complex components
- Dialogue systems involve complex interactions among these individually complex components
- Dialogue systems are becoming ubiquitous
- The model for most people is human-human interaction

Considerations for dialogue manager

- Prototyping
  - How easy is it to get a 0^{th} order iteration up and running?
  - What modules are included?
  - Are I/O specs standardized/easy to understand?
  - How easy is it to expand the system?
  - Are modules black boxes?
- Robustness
  - Are fallback mechanisms implemented?
  - Is there error catching?

Considerations for dialogue manager

- Expertise required
  - Is there a separate scripting language?
  - How is basic functionality (i.e., ASR, response generation) expanded?
  - How much computational linguistics, acoustic phonetics, signal processing, UI design is needed?

Approaches to building more complex SDS

- Architectures:
  - Information State Update model (University of Edinburgh)
  - Galaxy Communicator (MIT)
- Dialogue Management schemes:
  - Information State Update approach (University of Edinburgh)
  - Data-driven (MIT)

Commonalities among “advanced” architectures

- Unify sets of software servers/agents, each performing different task
- Control flow of information among servers
- Are rule-based at some level
- Have stores for global variables

Design considerations for research architectures

- Sequential rules vs. blackboard
- Unification of all HLT servers
- Common IO specs
- Plug-and-play

Open-Agent Architecture

- Allows integration of software agents for prototyping dialogue system
- Agents conform to conventions of framework
- Use common language for communication
- "Facilitator" mediates interaction among agents
- Facilitator maintains ordering constraints implicitly

Information-State Update Model

- Core: Dialogue Move Engine
  - Receives input from other agents (e.g., ASR)
  - Updates internal state to reflect new information
  - Calls other agents (e.g., TTS)
- Declarative representation of dialogue modelling
  - Specification of contents of dialogue
  - Datatypes for information state
  - Update rules for dealing with dynamic information
  - Control strategy

DIPPER: an implementation of the ISU model

- Update language independent of any particular programming language
- Incorporates many off-the-shelf OAA agents

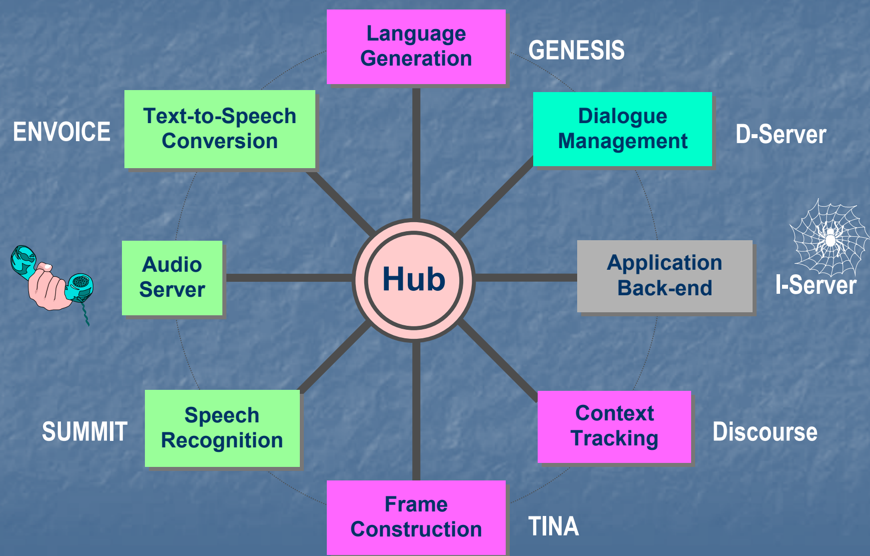
Galaxy Communicator

- Sequential rules
- Configuration specifically aimed at spoken dialogue systems
- Multiple servers interacting with one central hub

Basic components

- Hub
  - Keeps track of global state
  - Mediates interaction among servers
  - Controls logging, global parameters
- Servers
  - Stateless
  - Connect to hub via control file
- Token
  - Global store for attributes
  - Unless otherwise specified, attributes disappear with new turn

Galaxy-II Architecture



Control Strategy

- A set of ordered rules is a "program"
  - Simple syntax supports boolean and arithmetic tests applied to hub variables
  - All rules that apply are simultaneously executed
    - Relevant input variables are packaged into a frame and sent to target server
    - Frame is queued by hub when target server is busy
- Each program has a separate name
  - The "main" program controls processing for user queries
  - Other programs control module-to-module sub-dialogues and asynchronous I/O

Control Strategy (cont'd)

- Upon start-up, hub sends a "welcome" frame to each server
  - Server-specific initializations
- Hub polls continuously for new inputs or replies
- New inputs generate new tokens
- Tokens are processed according to program rules
- Replies modify existing tokens
- Tokens destroyed when no further rules apply
- Multiple users are managed via distinct sessions
  - Retain state for user's dialogue; e.g., language, domain, discourse context, etc.

Sample rule

RULE:	**:parseFrame & !:requestFrame → contextTracking**
RETRIEVE:	**:historyFrame**
IN:	**:parseFrame**
OUT:	**:requestFrame :historyFrame :domain**
STORE: :historyFrame	

- Boolean tests on attributes in global token
- IN and OUT keys specify specific attributes to retrieve from and store in token
- RETRIEVE and STORE used for attributes in global store
- Rules can also
  - Specify logging parameters (both into and out of operation)
  - Rename variables

Turn Management (the heart of Dialogue Management)

- Phases of turn management
- Making turn management domain independent
- Making turn management data-driven
  - Using data to determine what to say
  - Using data to determine concepts

Roles of dialogue management in information retrieval domains

- Resolve ambiguities
  - Ambiguous input constraint (e.g. Miami, Florida or Miami, Ohio)
  - Pragmatic considerations (e.g., too many flights to speak)
- Inform and guide user
  - Suggest subsequent sub-goals (e.g., what time?)
  - Offer dialogue-context dependent assistance upon request
  - Provide plausible alternatives if requested information unavailable
  - Initiate clarification sub-dialogues for confirmation
- Influence other system components
  - Adjust language model due to dialogue context
  - Adjust discourse history due to pragmatics: "Christmas" = Dec25
  - Set up context for system initiative: "where to?" = destination

Phases in dialogue management

- Verify input
  - Check confidence scores
  - Deal with system initiative

    What day will you be arriving?

    November 23rd. → *interpret as arrival date*
- Determine whether a query should be sent to the database
  - Have sufficient constraints been elicited from user?

    I need a hotel in Boston. → query for brand or location
  - Can the query be resolved from previous response?

    What is the address of the third one? → from previous res

Phases in dialogue management

- Retrieval
  - Construct frame for database query
  - Paraphrase database query frame
  - Connect to database and retrieve tuples

Phases in dialogue management

Input query ↓

Pre-retrieval → Retrieval → **Filtering** → Response construction

- Filter result from database, based on constraints from user

    Which one is cheapest? → find cheapest in database tuples

    I'd like a Sheraton. → filter database tuples for brand

- Order database result

    I've found three hotels near the airport. The Airport Hilton for $219.00, the Sheraton Logan for $230.00 and the Marriott at Logan for $249.00. → response list ordered by price

Phases in dialogue management

Input query ↓

Pre-retrieval → Retrieval → Filtering → **Response construction**

- Speak database tuples or summarize
- Add comments when necessary

    There is no Hyatt near the airport. There are two Hyatts in Boston …

- Add system initiatives and/or continuant prompt

    What city are you interested in?

    I have found three hotels …. Please select one.

- Provide help/meta-level responses

    You've been asking about hotels in Boston. You can now specify a brand of hotel or location in Boston.