Spoken Dialog Systems
  - Details of *Olympus* modules
  - Dialog Task Design
The Olympus Architecture

- Recognition Engine (AUDIOSERVER)
- Interpretation (PHOENIX)
- Interaction Manager (APOLLO)
- Synthesis (KALLIOPE)
- Generation (ROSETTA)
- Dialog Manager (RAVENCLAW)
- Backend

- Phone / Desktop
- Knowledge Source
Plan-based dialog manager

Task-independent engine
- core Olympus library
- manage dialog by executing task specification
- provides generic domain-independent behavior
  - Help, repeat, …
  - Confirmation, non-understandings…

Dialog Task Specification
- dialog plan
- interpretation context
RavenClaw Architecture

**Task-independent Dialog Engine**
- Manages dialog by executing the dialog task specification
- Provides many domain-independent conversational strategies
- Standard for most applications
  - No need to modify, just link shared library

**Dialog Task (Specification)**
- Captures all domain-specific dialog (task) logic using a hierarchical description
- Unique to each application
  - Must be created for each application
  - Links to dialog engine library
**Tree of dialog agents**
- **Terminals:** Inform, Request, Expect, Execute
- **Non-terminals / Dialog agency:** plans execution of child nodes

**Hierarchical Task Execution Network; each agent:**
- **Preconditions**
- **Success & failure criteria**
- **Trigger (focus) criteria**
- **Effects**
Sample Task Specification Code

```plaintext
// /Madeleine/GeneralFeel
DEFINE_AGENCY(CGGeneralFeel,
   DEFINE_CONCEPTS(
      STRING_USER_CONCEPT(general_feeling, none))
   DEFINE_SUBAGENTS(
      SUBAGENT(HowAreYou, CHowAreYou)
      SUBAGENT(Glad, CGlad)
      SUBAGENT(Sorry, CSorry))
   SUCCEEDS_WHEN(COMPLETED(Glad) || COMPLETED(Sorry)))

// /Madeleine/GeneralFeel/HowAreYou
DEFINE_REQUEST_AGENT(CHowAreYou,
   REQUEST_CONCEPT(general_feeling)
   GRAMMAR_MAPPING("![Yes]>good, ![FeelingGood]>good,"
   "![FeelingSoSo]>soso, ![FeelingBad]>bad"))

// /Madeleine/GeneralFeel/Glad
DEFINE_INFORM_AGENT(CGlad,
   PRECONDITION(C("general_feeling") == CString("good")
   PROMPT("inform glad_youre_good")
   ON_COMPLETION(FINISH(/Madeleine)))

// /Madeleine/GeneralFeel/Sorry
DEFINE_INFORM_AGENT(CSorry,
   PRECONDITION(C("general_feeling") != CString("good")
   PROMPT("inform sorry_youre_bad"))
```

RavenClaw Task Specification Language (RCTSL)

- *(Pseudo-)*declarative language
  - Defines concept types
  - Describes the task tree

- Set of C++ macros
  - Concept types and agents are classes
  - Can use pure C++ code if necessary
  - Need to be recompiled when modified
RCTSL Concepts

- **Concepts are effectively RCTSL variables**
  - Store values for later use and manipulation

- **Standard types**
  - String, integer and bool

- **User-defined types**
  - Structures and arrays

- **Two main categories:**
  - System concepts
    - Store internal values, database results, etc.
  - User concepts
    - Capture entities obtained from the user
How User Concepts get Values

◆ GRAMMAR_MAPPING directive

● Defines which grammar slot(s) from Phoenix are assigned to an expected concept

```plaintext
// /MyBus/PerformTask/GetQuerySpecs/RequestOriginPlace
DEFINE_REQUEST_AGENT( CRequestOriginPlace,
    REQUEST_CONCEPT(origin)
    PROMPT("request origin_place")
    GRAMMAR_MAPPING("[origin_place], ![Place]")
)
```

● Maps parsed value from grammar (slot [origin_place]) to concept origin
Specifying Binding Scope

- **Initiative can be controlled via binding scope**
  - System vs. Mixed initiative

- **Grammar mappings encode binding scope:**
  - Special character before grammar slot name
  - Strict (!): bind only when request agent is active
  - Open (@): bind always
  - Default (Ø): bind only when request agent’s subtask is active

```c
// /MyBus/PerformTask/GetQuerySpecs/RequestOriginPlace
DEFINE_REQUEST_AGENT( CRequestOriginPlace, 
  REQUEST_CONCEPT(origin)
  PROMPT("request origin_place")
  GRAMMAR_MAPPING("[origin_place], ![Place]")
}
```
RavenClaw Execution

Dialog Stack  Expectation Agenda

I: Welcome  E: LoadSymptoms  GeneralFeel
R: HowAreYou?  I: Glad  I: Sorry
I: Welcome  E: LoadSymptoms  GeneralFeel
R: HowAreYou?  I: Glad  I: Sorry

I: Welcome  E: LoadSymptoms  GeneralFeel
R: HowAreYou?  I: Glad  I: Sorry

chart  diagnostic

general_feeling

have_fever
RavenClaw Execution

Dialog Stack | Expectation Agenda
--- | ---
Madeleine

I: Welcome
E: LoadSymptoms
GeneralFeel
R: HowAreYou?
I: Glad
I: Sorry
Fever
Travel
Diagnose

chart
diagnostic
general_feeling
have_fever

I: Welcome
E: LoadSymptoms
GeneralFeel
R: HowAreYou?
I: Glad
I: Sorry

R: AskFever
E: MeasureTemp
I: InformFever
**RavenClaw Execution**

**Dialog Stack**

- Welcome
- Madeleine

**Expectation Agenda**

- chart
- diagnostic
- general_feeling
- have_fever

- GeneralFeel
  - HowAreYou?
  - Glad
  - Sorry

- Diagnose
  - AskFever
  - MeasureTemp
  - InformFever

- Fever
- Travel
Hi, this is Madeleine, the automated...
Hi, this is Madeleine, the automated...
Hi, this is Madeleine, the automated...
Hi, this is Madeleine, the automated...
Hi, this is Madeleine, the automated...

How are you feeling today?

Not so good, I think I have a fever

(soso)(not so good)

(fever)(I think I have a fever)
Hi, this is Madeleine, the automated...
How are you feeling today?

Not so good, I think I have a fever
[soso](not so good)
[fever](I think I have a fever)
Hi, this is Madeleine, the automated...
How are you feeling today?

Not so good, I think I have a fever
[not so good]
[fever]

Oh, I’m sorry to hear that...
Let me take your temperature...
RavenClaw – Other features

- **Transparently provides conversational skillset**
  - **Universal dialog mechanisms:**
    - Repeat, Quit, etc.
  - **Help:**
    - Help!, What can I say?
  - **Error handling:**
    - Explicit and implicit confirmations
    - Strategies for recovering from non-understandings

- **Dynamic dialog task generation**
- **Dynamic dialog control policy**
The Olympus Architecture

- **Recognition Engine (AUDIOSERVER)**
- **Interpretation (PHOENIX)**
- **Interaction Manager (APOLLO)**
- **Dialog Manager (RAVENCLAW)**
- **Synthesis (KALLIOPE)**
- **Generation (ROSETTA)**
- **Knowledge Source**
- **Phone / Desktop**
- **Backend**

- **Synth. Engine (SAPI/FLITE)**
- **Recog. Engine (SPHINX)**
Rosetta Language Generation

- Template- and stochastic-based language generation
  - Input: (act, object, {slot=value})
  - Output: text (tagged with concepts)
- Takes semantic output from the dialog manager, generates corresponding surface forms

```perl
# welcome to the system
"welcome" => "Welcome to RoomLine, the automated conference room."
    "reservation system."
# greet user
"greet_user" => ("Hello, <user_name>.",
    "Hi, <user_name>, good to hear from you again.")
# inform the user that the system has misunderstood the times (order)
"wrong_time_order" => sub {
    my %args = @_; 
    my $time_interval_as_string = get_wrong_time_interval_as_string(%args,
        "room_query.date_time.time");
    my $answer = "I'm sorry, I must have misunderstood the "
    "time you needed the room. "]
    $answer .= "I heard $time_interval_as_string. ;"
    return ["$answer So, let's see ... ",
        "$answer So, let's try this again ... ",
        "$answer So, let's try this once more ... "];
},
```
Good input and output language!

- List expected user utterances
  - Get several people to write example sentences, to improve coverage!
  - Use to design grammar for system understanding
- Write system prompts
  - Be concise!! Nice written language does not translate well into spoken language...
Designing a Dialog Task

Structure the task specification
- If it’s a tree-based system, draw out the tree!
- Define what information is needed/expected, and where

Typical information-giving systems tend to have similar structure
- **greet – do task – goodbye**
  - *do task: get info – process info – give answer*
S: Welcome to MyBus.
S: Which itinerary are you looking for?
U: I need to go to the airport.
S: Where are you leaving from?
U: Downtown.
S: Let me see.
S: There is a 28X leaving downtown at 10:15 AM. It will get to the airport at 11 AM.
S: You can say …
U: When is the next bus?
S: There is a 28X leaving downtown at 10:45 AM. It will get to the airport at 11:30 AM.
<table>
<thead>
<tr>
<th>S: Welcome to MyBus.</th>
<th>Open dialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>S: Which itinerary are you looking for?</td>
<td>Get user query</td>
</tr>
<tr>
<td>U: I need to go to the airport.</td>
<td>Process user query</td>
</tr>
<tr>
<td>S: Where are you leaving from?</td>
<td>Present and discuss results</td>
</tr>
<tr>
<td>U: Downtown.</td>
<td></td>
</tr>
<tr>
<td>S: Let me see.</td>
<td></td>
</tr>
<tr>
<td>S: There is a 28X leaving downtown at 10:15 AM. It will get to the airport at 11 AM.</td>
<td></td>
</tr>
<tr>
<td>S: You can say ...</td>
<td></td>
</tr>
<tr>
<td>U: When is the next bus?</td>
<td></td>
</tr>
<tr>
<td>S: There is a 28X leaving downtown at 10:45 AM. It will get to the airport at 11:30 AM.</td>
<td></td>
</tr>
</tbody>
</table>
S: Welcome to MyBus.
S: Which itinerary are you looking for?
U: I need to go to the airport.
S: Where are you leaving from?
U: Downtown.
S: Let me see.
S: There is a 28X leaving downtown at 10:15 AM. It will get to the airport at 11 AM.
S: You can say …
U: When is the next bus?
S: There is a 28X leaving downtown at 10:45 AM. It will get to the airport at 11:30 AM.
S: Welcome to MyBus.
S: Which itinerary are you looking for?
U: When is the next bus from downtown to the airport?
S: Let me see. There is a 28X leaving downtown at 10:15 AM. It will get to the airport at 11 AM.
S: You can say …
U: Goodbye.
S: Thank you for using MyBus. Goodbye.