Lecture 4: Frame-based and Information-state based Approaches

David Traum

traum@ict.usc.edu

http://www.ict.usc.edu/~traum
Outline

• Frame-based approach
  - Example systems: MIT

• Frame+agenda
  - CMU

• Information-state approach
  - Trindikit
  - Other kits

• Example information-based theories & systems
  - EDIS
Transaction Dialogues

- User has a request
- System needs info from user to process request
- Dialogue proceeds as:
  - User specifies request
  - System gathers necessary info
    - Q&A
      - Spontaneous assertion from user
  - System looks up information & provides response
Frame-based Approach

• Also called form-based (MIT)
• Central data structure is frame with slots
  - DM is monitoring frame, filling in slots
• Used for transaction dialogues
• Generalizes finite-state approach by allowing multiple paths to acquire info
• Frame:
  - Set of information needed
  - Context for utterance interpretation
  - Context for dialogue progress
• Allows mixed initiative
Example: MIT Wheels system

- Domain: searching used car ads
- Transaction domain + constraint satisfaction
- No slots are mandatory,
  - try to find the best set of matches
  - Try to find an appropriate # of matches
Example: MIT Jupiter System (1)

- Retrieval of weather forecast domain
  - Multiple sources
  - Content processing
  - Information on demand
  - Context

- 1-888-573-8255
MIT Jupiter System (2)

- Uses Galaxy architecture
  - SUMMIT ASR
    - 2000 word vocabulary, 1-9% OOV
  - TINA NL understanding
    - Creates semantic frames from text
    - Used for both query understanding (user)
    - Content understanding (web-based weather text)
  - GENESIS generation
    - User text
    - SQL queries
    - Keyword-value
- Dialogue control table
  - Conditions for operations
  - context
Problems with Frames

- Not easily applicable to complex tasks
  - May not be a single frame
  - Dynamic construction of information
  - User access to “product”
Agenda + Frame (CMU Communicator)

- **Product:**
  - hierarchical composition of frames

- **Process:**
  - Agenda
    - Generalization of stack
    - Ordered list of topics
    - List of handlers
Example: CMU Communicator System
The Information State Approach to Dialogue Modelling: Some Results of the TRINDI Project

David Traum

USC Institute for Creative Technologies
traum@ict.usc.edu
TRINDI Project

- Task-Oriented Instructional Dialogue
- European Union Telematics, 2yr project (1998-2000)
- ~15 Researchers
- Consortium: U Gothenburg, U Edinburgh, U Saarlandes, SRI Cambridge, Xerox
Motivating Problems

• Dialogue theories are largely incomparable
  - despite often similar intended coverage
  - e.g., motivation for answering questions:
    - cooperativity vs. obligations vs. QUD structure
    - Heterogeneous building blocks

• Large gap between dialogue models in systems and broad-coverage theories

• Dialogue systems are hard to build
  - despite rapid progress in ASR, TTS, NLP
  - hard to convert systems to new domains
  - insufficient attention to `theoretical’ concerns
Deficiencies of Previous Dialogue Theories

• Inappropriate for direct implementation
  - Some aspects too vague
    • e.g., Relevance Theory (a la Sperber and Wilson)
  - some aspects too complex for efficient computation
    • e.g., Implicit Belief using Modal Predicate Logic
• Hard to evaluate/compare with other theories
  • even when covering same dialogue phenomena
  • Heterogeneous building blocks
    - How to combine, e.g., mentalistic and structural
Deficiencies of Current Dialogue Systems

- Software engineering challenge
  - combining heterogeneous sub-systems
- Domain/Task specific design
  - little carried over to next system
- Insufficient attention to dialogue structure
  - Dialogue usually conceived as FSM
    - inflexible interaction
    - does not scale to large tasks
Partial Solution: Dialogue Toolkits

- Software Integration (OAA, Trains/Trips, Verbmobil)
- FSM Dialogue Kits (Nuance, OGI, ...)
- Slot-Filling (Phillips)
- Current Development Kits:
  - Utterance-based (DARPA Communicator)
  ⇒ Information-based (TrindiKit)
Approach to Problems

- Information State approach to formalizing theories of dialogue modelling
- Dialogue Move Engine (TrindiKit) for implementing a dialogue modelling theory
- Example implementations
- Comparative experimentation, enhancements, & evaluation
Information State Theories of Dialogue

• Statics
  - Informational components (functional spec)
    • e.g., QUD, common ground, dialogue history, ...
  - formal representations (accessibility)
    • e.g., lists, records, DRSes, ...

• Dynamics
  - dialogue moves
    • abstractions of i/o (e.g., speech acts)
  - update rules - atomic updates
  - update strategy - coordinated application of rules
Sample GoDiS information state

PRIVATE =

AGENDA = { findout(?return) }
|      | findout(?λx.month(x)) |
|      | findout(?λx.class(x)) |
|      | respond(?λx.price(x)) |

PLAN =

BEL = {}

TMP = (*same as SHARED*)

SHARED =

COM =
|             | dest(paris) |
|             | transport(plane) |
|             | task(get_price_info) |

QUD = < λx.origin(x) >

LM = { ask(sys, λx.origin(x)) }
Sample GoDiS update rule

- integrateAnswer
  
  pre: \[
  \begin{align*}
  &\text{in}(\text{SHARED.LM}, \text{answer}(\text{usr}, A)) \\
  &\text{fst}(\text{SHARED.QUD}, Q) \\
  &\text{relevant-answer}(Q, A) \\
  \end{align*}
  \]

  eff: \[
  \begin{align*}
  &\text{pop}(\text{SHARED.QUD}) \\
  &\text{reduce}(Q, A, P) \\
  &\text{add}(\text{SHARED.COM}, P) \\
  \end{align*}
  \]
Dialogue Move Engine

- Handles Dialogue Management tasks:
  - consumes observed dialogue moves
  - updates information state
  - produces new dialogue moves to be performed
- Can be implemented as:
  - Update (&Selection) Rules
  - Update Algorithm
TrindiKit

- Architecture based on information states
- Modules (dialogue move engine, input, interpretation, generation, output etc.) access the information state
- Resources (databases, lexicons, domain knowledge etc.)
TrindiKit

- Input
- Interpret
- Control
- Update
- Select
- DME
- Generate
- Output

Information State

- Resource
- Resource
TrindiKit Features

- Explicit information state data-structure
  - makes systems more transparent
  - closer to dialogue processing theory
  - easier comparison of theories
- modularity for simple and efficient reconfiguration and reusability
- rapid prototyping
TrindiKiT Includes

- A library of datatype definitions
  - conditions and operations
- facilities for writing update rules and algorithms
- tools for visualizing information state
- debugging facilities
- A library of basic ready-made modules for i/o, interpretation, generation, etc.
- Resource interfaces
Building a TrindiKit system

Build or select from existing components:
- Type of information state (DRS, record, …)
- A set of dialogue moves
- Information state update rules,
- DME Module algorithm(s), including control algorithm
- Resources: databases, grammars, plan libraries etc., or external modules
Building a system

- Domain knowledge (resources)
- Dialogue theory (IS, rules, moves etc)
- Software engineering (basic types, control flow)

→ Domain-specific system
→ Domain-independent DME
→ TRINDIKIT
TrindiKit Systems

- GoDiS (Larsson et al) - information state: Questions Under Discussion
- MIDAS - DRS information state, first-order reasoning (Bos & Gabsdil, 2000)
- EDIS - PTT Information State, (Matheson et al 2000)
- SRI Autoroute - information state based on Conversational Game Theory (Lewin 2000) Robust Interpretation (Milward 1999)
System Comparisons

- Cross-IS Theories: SRI vs. EDIS on AutoRoute Dialogues
- Different formalizations: PTT using DRSes or Records
- Different Update strategies:
  - GoDiS with or without plan accommodation
  - Midas using different grounding strategies
- Different Languages, Tasks, and interactivity
  - GoDiS: English vs. Swedish
  - GoDiS: AutoRoute vs. Travel Agent
  - IMDIS: dialogue vs. text
Potential Impact

- Better development environment for formal dialogue theories
  - easy testing/revision of theories
  - comparison across theories
- Closer integration of theories and systems
- Better dialogue system development
  - Information state vs. dialogue state
  - extension to other domains
Post-Trindi Applications

- Siridus Project (EU 2000-)
  - Command and negotiative dialogues
  - Spanish
  - GoDiS, SRI
- IBL for Mobile Robots (U Edinburgh)
  - Midas
- Tutoring Electricity (U Edinburgh)
  - EDIS
Successor Toolkits

- TrindiKit revisions
- Dipper
- Midiki
EDIS SYSTEM

- Uses PTT theory
- Trindikit implementation
- Autoroute domain
PTT Informational Components

- Separate Views for System and User (System assumptions about User)

- Private, Public, and Semi-public components of View captures *grounding* process (Clark & Schaefer '87)
  - GND represents common ground
  - set of DUs represent partitioned semi-public information introduced but not (yet) grounded
  - UDUs structure accessible ungrounded DUs

- (Semi-)Public Information includes:
  - public events
  - social commitments of participants

- Private Information includes
  - Intentions
  - Beliefs
EDIS Formalization of Information Components

• Record (AVM) for Views, with fields for each dialogue participant:
  - GND: PT-Rec
    Public Information
  - UDUS: list of accessible DU IDs
  - CDU (DU-ID, PT-Rec)
    current Discourse Unit
  - PDU (DU-ID, PT-Rec)
    penultimate Discourse Unit
  - INT: list of intended actions

• PT-REC contains:
  - DH: list of dialogue acts
    Dialogue History of performed dialogue acts
  - OBL: list of action types
    Obligations of participants to perform actions
  - SCP: list of states
    Social Commitments of agents to Propositions
  - COND: list of implications
    relevant conditional anticipated effects
PTT Information State

\[
\begin{align*}
G & : PT-R \\
CDU & : \begin{bmatrix}
C & : & PT-R \\
ID & : & DU-ID
\end{bmatrix} \\
PDU & : \begin{bmatrix}
C & : & PT-R \\
ID & : & DU-ID
\end{bmatrix} \\
UDUs & : \text{List(DU-ID)} \\
INT & : \text{List(Action)}
\end{align*}
\]

\[
PT-R : \begin{bmatrix}
DH & : & \text{List(Action)} \\
OBL & : & \text{List(Action)} \\
SCP & : & \text{List(Prop)} \\
COND & : & \text{List(Action)}
\end{bmatrix}
\]
EDIS Dialogue Moves

• Forward-looking
  - assert(dp,Prop)
  - check(dp, Prop)
  - direct (dp,act-type)
  - info-request(dp,Q)

• Backward Looking
  - Address(dp,act)
    • accept
    • agree
    • answer
  - Understanding Act
    • Acknowledge(dp,DU-ID)
Update Strategy

- Deliberation (produce new intentions)
- Acting on intentions (produce output dialogue moves)
- Update based on an observed utterance
  1. Create a new DU and push it on top of UDUs.
  2. Perform updates for backwards grounding acts.
  3. For other types, record in cdu.dh and apply the update rules for act class.
  4. Apply inference update rules to all parts of the IS which contain newly added acts.
Update Rules

- effects of observed dialogue acts
  - formalized in terms of social commitments

- inference
  - Obligation Resolution
  - Conditional Resolution
  - Intention Resolution

- Deliberation
  - adopting new intentions
<table>
<thead>
<tr>
<th>act</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID:2, <strong>ack</strong>(DP,DU1)</td>
<td>peRec(w.Gnd,w.pdu.tognd)</td>
</tr>
<tr>
<td></td>
<td><strong>remove</strong>(DU1,UDUS)</td>
</tr>
<tr>
<td>ID:c, <strong>forward-looking-act</strong>(DP)</td>
<td>push(obl,u-act(o(DP),CDU.id))</td>
</tr>
<tr>
<td></td>
<td><strong>accomplished via rule resolution</strong></td>
</tr>
<tr>
<td>ID:2, <strong>agree</strong>(DP,ID2)</td>
<td>push(scp,scp(DP,P(ID2))))</td>
</tr>
<tr>
<td>effect</td>
<td>push(scp,ans(DP,Q(ID2),P(ID2)))</td>
</tr>
<tr>
<td>ID:2, <strong>assert</strong>(DP,PROP)</td>
<td>push(scp,scp(DP,PROP))</td>
</tr>
<tr>
<td>effect</td>
<td>push(obl,address(o(DP),ID))</td>
</tr>
<tr>
<td></td>
<td>push(cond,agree(o(DP),ID) →</td>
</tr>
<tr>
<td></td>
<td>scp(o(DP),PROP))</td>
</tr>
<tr>
<td>ID:1, <strong>assert</strong>(DP,PROP)</td>
<td>push(cond,accept(o(DP),ID) →</td>
</tr>
<tr>
<td>effect</td>
<td>scp(o(DP),PROP))</td>
</tr>
<tr>
<td>ID:2, <strong>check</strong>(DP,PROP)</td>
<td>push(obl,address(o(DP),ID))</td>
</tr>
<tr>
<td>effect</td>
<td>push(cond,accept(o(DP),ID) →</td>
</tr>
<tr>
<td></td>
<td>scp(DP,PROP))</td>
</tr>
<tr>
<td>ID:2, <strong>direct</strong>(DP,Act)</td>
<td>push(obl,address(o(DP),ID))</td>
</tr>
<tr>
<td>effect</td>
<td>push(cond,accept(o(DP),ID) →</td>
</tr>
<tr>
<td></td>
<td>obl(o(DP),Act))</td>
</tr>
<tr>
<td>ID:2, <strong>info_request</strong>(DP,Q)</td>
<td>push(obl,address(o(DP),ID))</td>
</tr>
<tr>
<td>effect</td>
<td>push(obl,address(o(DP),ID))</td>
</tr>
</tbody>
</table>
Deliberation Factors

- obligations
  - to perform understanding acts
  - to address previous dialogue acts
  - to perform other actions

- potential obligations that would result if another act were performed, as represented in the cond field (or CDU.OBL)

- insufficiently understood dialogue acts with a 1 confidence level in cdu.dh

- intentions to perform complex acts
Deliberation Rules

1. Grounding:
   OBL U·act, everything in CDU understood
   ⇒ ack(W,CDU)

2. Address:
   OBL address act
   ⇒ accept, agree, or answer

3. Anticipatory Planning:
   INT act1 ∧ COND act1 → OBL act2
   ⇒ act2 add an intention to perform an action

4. SubGoal: Int(act1) ∧ NextSubact(Act1,Act2)
   ⇒ Act2
   (a) check CDU.DH:1
   (b) info-request
Sample Autoroute Dialogue

WIZARD
[1]: How can I help you?
[3]: Where would you like to start?
[5]: Great Malvern?
[7]: Where do you want to go?
[9]: Edwinstowe in Nottingham?
[11]: When do you want to leave?
[13]: Leaving at 6 p.m.?
[15]: Do you want the quickest or the shortest route?
[17]: Please wait while your route is calculated.

CALLER
[2]: A route please
[4]: Malvern
[6]: Yes
[8]: Edwinstowe
[10]: Yes
[12]: Six pm
[14]: Yes
[16]: Quickest
InfoState after [2]: A route please

OBL: \[\text{understandingAct}(W,DU3)\]
    \[\text{address}(C,CA2)\]

GND:

DH:
\[\text{CA3: C2, acknowledge}(C,DU2)\]
\[\text{CA2: C2, info_request}(W,\text{?helpform})\]

SCP: < >
COND: < >

UDUS: \(<DU3>\)

PDU:

TOGND:

W:
ID: DU2

CDU:

TOGND:

ID: DU3

INT:
\[\text{info_request}(W,\text{?start})\]
\[\text{giveroute}(W)\]
\[\text{accept}(W,CA6)\]
\[\text{acknowledge}(W,DU3)\]

C: \[\text{INT: <getroute(C)> }\]
InfoState after [4]: Malvern, prompting check

```
OBL:  \{ giveroute(W) \\
   \quad understandingAct(W,DU5) \\
   \quad address(C,CA8) \\
\}

GND: \{ CA10: C2, acknowledge(C,DU4) \}

DH: \{ CA9: C2, accept(W,CA6) \\
\quad CA8: C2, info_request(W,?start) \}

SCP: < >

COND: < >

UDUS: <DU5>

W: PDU: TOGN1:  
   \{ OBL: <address(C,CA8)> \}
   \{ DH: \quad CA9: C2, accept(W,CA6) \}
   \{ SCP: < > \}
   \{ COND: < > \}

ID: DU4

CDU: TOGN1:  
  \{ OBL: < > \}
  \{ DH: \quad CA12: C2, answer(C,CA8,CA11) \}
  \{ SCP: < > \}
  \{ COND: < > \}

ID: DU5

INT: \{ check(W,start(malvern)) \}

\quad acknowledge(W,DU5) \}

\quad giveroute(W) \}

C: \{ INT: <getroute(C)> \}
```
InfoState after [5]: Great Malvern?

OBL: \<understandingAct(C,DU6)\>
    \<giveroute(W)\>

CA13: C2, \acknowledge(W,DU5)\>

CA12: C2, \answer(C,CA8)\>

CA11: C1, \assert(C,\text{start(malvern)})\>

SCP: < >

COND: < >

UDUS: <DU6>

W:

OBL: < >

CA12: C2, \answer(C,CA8,CA11)\>

CA11: C1, \assert(C,\text{start(malvern)})\>

TOGND:

SCP: < >

COND: < >

ID: DU5

PDUS:

OBL: < >>address(C,CA14)\>

TOGND:

DH: < >>CA14: C2, \check(W,\text{start(malvern)})\>

SCP: < >

COND: < >>\text{agree(C,CA14)} -> scp(W,\text{start(malvern)})\>

ID: DU6

CDU:

INT: < >>giveroute(W)\>

C: [INT: < >>getroute(C)\>]
InfoState after [7]: Where do you want to go?

```
OBL: 〈understandingAct(C,DU8)〉
giveroute(W)

DH: 〈CA17: C2, acknowledge(W,DU7)〉
CA16: C2, agree(C,CA14)

SCP: 〈scp(C,start(malvern))〉
scp(W,start(malvern))

COND: < >

UDUS: <DU8>

W: 〈OBL: < >〉
DH: 〈CA16: C2, agree(C,CA14)〉

PDU: 〈TOGND: SCP: 〈scp(C,start(malvern))〉
COND: < >〉
ID: DU7

ID: DU7

CDU: 〈TOGND: SCP: < >
COND: < >〉
ID: DU8

INT: 〈giveroute(W)〉

C: 〈INT: <getroute(C)> [ ]
```