Extendable Dialog Script Description Language for Natural Language User Interfaces

13/4/16

Yahoo Japan Research
Kiyoshi Nitta
knitta@yahoo-corp.jp
Introduction
I. Utilize Emerging Large-Scale Ontology Resources

II. Two Possible Approaches for Accumulating Large-Scale Dialog Scripts

III. Challenges

IV. Contributions
A) Artificial General Intelligence and Narrow AI
B) Rapidly Growing LOD Project Community
C) Breakthrough in AGI Side
D) Large-Scale Dialog Scripts: An Application
Artificial General Intelligence and Narrow AI

Artificial General Intelligence

- Brain Emulation
- Mind Emulation
- Artificial Life
- Social Intelligence
- Pragmatic Logic
- 5th Generation Computing (Japan)
- Human Made Knowledge Base (Cyc)
- Semantic Web

Narrow AI

- Expert System
- Natural Language Processing
- Speech Recognition
- Image Recognition
- Machine Learning
Rapidly Growing LOD Project Community

2009

(Bizer et al. 2009)

2011

Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/
Breakthrough in AGI Side

Manually constructed knowledge base

Collaboratively accumulated knowledge base

Photo by inmymemory

Photo by inmymemory
Large-Scale Dialog Scripts: As An Application

Large-Scale Knowledge Base

Information Retrieval

Question Answering

Conversational Agents

Proactive Agents

Watson

WolframAlpha

Evi (TrueKnowledge)

Siri
Two Possible Approaches for Accumulating Large-Scale Dialog Scripts

A) Machine Learning Approach: Use Operation or Conversation Log as Training Data

B) UGC Approach: Accumulate Dialog Scripts as UGC
Challenges

A) Extensibility / Flexibility
   - Ontology Extensibility
   - Functional Extensibility

B) Heavy Load Tolerance

C) Efficiency
Contributions

A) design of a flexible dialog script language (GLDS)
B) implementation of GLDS execution engines
C) evaluation of conversational operations for modifying dialog scripts
Framework
I. Graph Structure and Semantics
II. (Structure Required for) Dialog Script Tracer
III. Graphical Language for Dialog Script (GLDS)
A) Element Sets
B) Three layers of Semantics
C) Base Semantics (Existence of Relations)
D) Interim Semantics (Class-Instance Relations)
E) Top Semantics (Semantics of Instances)
F) Base Graph Structure of the Framework
Element Sets

\[ E_v = \left\{ e(e_s, e_d) \mid e_s = \text{null}, e_d = \text{null} \right\} \]

vertex set

\[ E_{hoe} = \left\{ e(e_s, e_d) \mid e_s \in E, e_d = \text{null} \right\} \]

head open edge set

\[ E_{toe} = \left\{ e(e_s, e_d) \mid e_s = \text{null}, e_d \in E \right\} \]

tail open edge set

\[ E_e = \left\{ e(e_s, e_d) \mid e_s, e_d \in E \right\} \]

edge set

\[ E = E_v \cup E_{hoe} \cup E_{toe} \cup E_e \]

whole element set
Three layers of Semantics

(1) Base Semantics
- relationship

(2) Interim Semantics
- class → instance

(3) Top Semantics
- class → instance
  - inherit
Edge element $e_e$ means that element $e_s$ has some kind of relationship to element $e_d$. 
Data element $e_d$ is an instance of class element $e_c$.

apriori classes: \[ E_{\text{apriori}} = \{ v_{\text{class}}, v_{\text{ins}} \} \]

instance relation set: \[ e \in E_{\text{ins}} \]

class set: \[ e_c \in E_{\text{class}} \]
Instance elements $e_{d1}, e_{d2}, e_{d3}, \ldots, e_n$ inherit the meaning from class element $e_c$.

Data set: $e_{d1}, e_{d2}, e_{d3}, \ldots, e_n \in E_{data}$
Base Graph Structure of the Framework

\[ \text{ADG}_F = (E_{apriori}, E_{ii}, E_{ins}, E_{class}, E_{data}) \]

- \( E_{apriori} \): apriori classes: “class” and “instance”
- \( E_{ii} \): edge set that indicates instance edges
- \( E_{ins} \): instance relation set
- \( E_{class} \): class set
- \( E_{data} \): data set
A) Must Classes
B) Context Vertex
C) Trigger Vertex
D) Action Vertex
E) Simple Example
Must Classes

\[ V_{\text{class}} \]

- \( C_{\text{srl}} \)
- \( C_{\text{srk}} \)
- \( C_{\text{rel}} \)
- \( C_{\text{rm}} \)
- \( C_{\text{sel}} \)

<table>
<thead>
<tr>
<th>class</th>
<th>type</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRL</td>
<td>vertex</td>
<td>root of dialog script</td>
</tr>
<tr>
<td>SRK</td>
<td>vertex</td>
<td>matching keyword</td>
</tr>
<tr>
<td>REL</td>
<td>edge</td>
<td>transitional relation</td>
</tr>
<tr>
<td>RM</td>
<td>vertex</td>
<td>reply message</td>
</tr>
<tr>
<td>SEL</td>
<td>vertex</td>
<td>select action</td>
</tr>
</tbody>
</table>
A dialog script must have a root instance element of SRL class.
A context vertex must be connected with at least a trigger vertex.
A trigger vertex must be connected with at least an action vertex.
Simple Example

Trigger vertexes and Action vertexes are connected by instance elements of REL class.
A) Class Definitions

B) Transitional Relation Modification

C) Variables
## Class Definitions

### Simple reply

- SM: Edge selection message relation

### Dialog controller access

- WCK: Vertex match a wild card keyword
- CFM: Vertex confirm an element
- CFA: Vertex confirm an array
- FMT: Edge format a relation
- NOP: Vertex no operation (dummy)
- SVC: Vertex save the context
- SLS: Vertex save the last selection
- SLR: Vertex save the last reply
- SVT: Vertex save a token
- QUI: Vertex quit

### Transitional relation modifier

- SUC: Edge success attribute
- FAI: Edge failure attribute
- IMM: Edge immediate attribute

### Variable argument

- VAR: Edge first argument
- VR2: Edge second argument
- VR3: Edge third argument

### Variable names

- CTX: Edge variable (context)
- LSK: Edge variable (last selected keyword)
- LRP: Edge variable (last reply)
- MSG: Edge variable (message)

### Variable operations

- LET: Vertex set literal
- COP: Vertex copy
- ISN: Vertex check null
- CLR: Vertex clear

### GLDS modification

- CSMS: Vertex search selection message
- CSMD: Vertex delete selection message
- CSMP: Vertex replace selection message
Transitional Relation Modification

$V_{class}$

$C_{rel}$

$C_{suc}$

$C_{fai}$

$C_{imm}$

$V_1$ $V_2$ $V_3$ $V_4$

$s$ $i$

$f$ $i$
Variables

\( V_{\text{class}} \)

- \( c_{\text{ses}} \)
- \( c_{\text{ctx}} \)
- \( c_{\text{lsk}} \)
- \( c_{\text{var}} \)
- \( c_{\text{vr2}} \)

\( s_{e1} \)

\( v_1 \)

\( \langle \text{ctx} \rangle \) \( \rightarrow \) \( v_1 \) \( \rightarrow \) \( \langle \text{lsk} \rangle \)
Example and Discussion
Example and Discussion

I. “Change Selection Message” Example
II. (Dynamic) Extensibility
“Change Selection Message” Example

A) “Change Selection Message” Dialog Script
B) Target Dialog Script
C) Saving Context and User Input
D) Confirmation Loop
E) Deleting Operation
F) Changing Operation
"Change Selection Message" Dialog Script
Target Dialog Script

User: Any news?
Agent: Source?
English, Japanese.

No news is good news!

Visit news.yahoo.com.
Visit headlines.yahoo.co.jp.
User: Change selection message
Agent: Specify a selection message to change.
User: Source
Agent: Do you want to change this selection message? selection message: `Source?'
yes, no, abandon
User: Yes
Deleting Operation

Agent: Do you want to delete it?
     yes, no, abandon.
User: No
A: Specify a new selection message.
U: What source do you have in mind?
A: Do you want to replace the selection message with this? selection message: 'What source do you have in mind?' yes, no, abandon.
U: Yes
A: Replaced.

m25: What source do you have in mind?
(Dynamic) Extensibility

A) Generations of GLDS
B) Difference between GLDS1 and GLDS2
C) Safe Extension of GLDS
D) Incremental Generation of Dialog Scripts
Generations of GLDS

GLDS_2

GLDS_1

GLDS_0

- devided
- editable
- basic
Difference between GLDS1 and GLDS2

**GLDS_1**
Write many similar codes.

```
kw_a  kw_b  kw_c  ...
  |
  |  csm_1  adt_1  adr_1  ...
  |
  |  c_csm  c_adt  c_adr  ...
```

**GLDS_2**
Reuse common parts.

```
kw_a  kw_b  kw_c  ...
  |
  |  svc_1  svc_2  svc_3  ...
  |
  |  svt_1  svt_2  svt_3  ...
```

Copyright (C) 2013 Yahoo Japan Corporation. All Rights Reserved.
Safe Extension of GLDS

\[ V_{\text{class}} \rightarrow C_{\text{srl}} \rightarrow \text{root} \]

\[ E_{\text{class}} \]

\[ \text{GLDS}_1 \]

\[ \text{GLDS}_2 \]

\[ GLDS_1 \quad GLDS_2 \quad E_{\text{data}} \]
Following editing commands were developed on GLDS$_2$ for incrementally generating GLDS$_0$ dialog scripts.

a) Register topic and reply  
b) Register additional reply  
c) Change reply to selection  
d) Register additional selection message  
e) Register additional keyword

These commands can also modify GLDS$_2$ dialog scripts, while the command set is not enough for generating full-spec GLDS$_2$ dialog scripts from scratch.
Conclusion
Conclusion

A) design of a flexible dialog script language (GLDS) and its base framework

B) implementation of GLDS execution engines
   - C++ version and Erlang version

C) evaluation of conversational operations for modifying dialog scripts
   - developed class definitions and dialog script commands for continuous three generations of GLDS
   - confirmed that developed GLDS$_2$ based dialog commands are enough to generate GLDS$_0$ dialog scripts from scratch