A Knowledge-based Methodology

for Building a Conversational Chatbot as an Intelligent Tutor

Xavier F. C. Sánchez-Díaz Gilberto Ayala-Bastidas Pedro Fonseca-Ortiz L. Garrido

Departamento de Computación Regional Norte
Escuela de Ingeniería y Ciencias
Tecnológico de Monterrey

MICAI 2018
October 25th 2018

The boom of conversational agents

Introduction

A Chatbot (or conversational agent) is a program designed to hold conversations with users using natural language.

Nearly half of the online interactions between 2007 and 2015 involved a chatbot[1].

Their use have been documented in a variety of contexts, including education and commerce.

One example is the development of Jill Watson as an intelligent tutor in Georgia Tech, for its artificial intelligence MOOC[2].

[1] Tsvetkova et al. 2017.

[2] Goel and Polepeddi 2016.

The boom of conversational agents

Introduction

A Chatbot (or conversational agent) is a program designed to hold conversations with users using natural language.

Nearly half of the online interactions between 2007 and 2015 involved a chatbot[1].

Their use have been documented in a variety of contexts, including education and commerce.

One example is the development of Jill Watson as an intelligent tutor in Georgia Tech, for its artificial intelligence MOOC[2].

[1] Tsvetkova et al. 2017.

[2] Goel and Polepeddi 2016.

The boom of conversational agents

Introduction

A Chatbot (or conversational agent) is a program designed to hold conversations with users using natural language.

Nearly half of the online interactions between 2007 and 2015 involved a chatbot[1].

Their use have been documented in a variety of contexts, including education and commerce.

One example is the development of Jill Watson as an intelligent tutor in Georgia Tech, for its artificial intelligence MOOC[2].

[1] Tsvetkova et al. 2017.

[2] Goel and Polepeddi 2016.

The boom of conversational agents

Introduction

A Chatbot (or conversational agent) is a program designed to hold conversations with users using natural language.

Nearly half of the online interactions between 2007 and 2015 involved a chatbot[1].

Their use have been documented in a variety of contexts, including education and commerce.

One example is the development of Jill Watson as an intelligent tutor in Georgia Tech, for its artificial intelligence MOOC[2].

[1] Tsvetkova et al. 2017.

[2] Goel and Polepeddi 2016.

Tools and frameworks

Introduction

Many frameworks exist to develop and implement chatbots:

- IBM's *Watson Assistant*
- Microsoft's *Bot Framework*
- Facebook's *Wit.ai*
- Google's *DialogFlow*
- Amazon's *Lex*

With so many tools available, building and deployment of a chatbot may look fairly simple.

Tools and frameworks

Introduction

Many frameworks exist to develop and implement chatbots:

- IBM's *Watson Assistant*
- Microsoft's *Bot Framework*
- Facebook's *Wit.ai*
- Google's *DialogFlow*
- Amazon's *Lex*

With so many tools available, building and deployment of a chatbot may look fairly simple.

Tools and frameworks

Introduction

Many frameworks exist to develop and implement chatbots:

- IBM's *Watson Assistant*
- Microsoft's *Bot Framework*
- Facebook's *Wit.ai*
- Google's *DialogFlow*
- Amazon's *Lex*

With so many tools available, building and deployment of a chatbot may look fairly simple.

Tools and frameworks

Introduction

Many frameworks exist to develop and implement chatbots:

- IBM's *Watson Assistant*
- Microsoft's *Bot Framework*
- Facebook's *Wit.ai*
- Google's *DialogFlow*
- Amazon's *Lex*

With so many tools available, building and deployment of a chatbot may look fairly simple.

Tools and frameworks

Introduction

Many frameworks exist to develop and implement chatbots:

- IBM's *Watson Assistant*
- Microsoft's *Bot Framework*
- Facebook's *Wit.ai*
- Google's *DialogFlow*
- Amazon's *Lex*

With so many tools available, building and deployment of a chatbot may look fairly simple.

Tools and frameworks

Introduction

Many frameworks exist to develop and implement chatbots:

- IBM's *Watson Assistant*
- Microsoft's *Bot Framework*
- Facebook's *Wit.ai*
- Google's *DialogFlow*
- Amazon's *Lex*

With so many tools available, building and deployment of a chatbot may look fairly simple.

Tools and frameworks

Introduction

Many frameworks exist to develop and implement chatbots:

- IBM's *Watson Assistant*
- Microsoft's *Bot Framework*
- Facebook's *Wit.ai*
- Google's *DialogFlow*
- Amazon's *Lex*

With so many tools available, building and deployment of a chatbot may look fairly simple.

Truth is...

Introduction

Providing the chatbot with suitable information to be able to work as an **educational tutor** could be difficult.

Information on how to design the tutor **is scarce** and scattered across blog entries and articles **focusing on implementation**.

Current frameworks only deal with implementation. A methodology for **knowledge abstraction and organization** is essential.

Truth is...

Introduction

Providing the chatbot with suitable information to be able to work as an **educational tutor** could be difficult.

Information on how to design the tutor **is scarce** and scattered across blog entries and articles **focusing on implementation**.

Current frameworks only deal with implementation. A methodology for **knowledge abstraction and organization** is essential.

Truth is...

Introduction

Providing the chatbot with suitable information to be able to work as an **educational tutor** could be difficult.

Information on how to design the tutor **is scarce** and scattered across blog entries and articles **focusing on implementation**.

Current frameworks only deal with implementation. A methodology for **knowledge abstraction and organization** is essential.

Proposed methodology

Our approach

A methodology for designing intelligent tutors that can be implemented on commercially available frameworks.

It considers two main phases:

Knowledge Modeling describes how knowledge is represented and stored in the knowledge base (KB).

Conversation Flow deals with the lexicon used by the tutor and the order in which ideas are presented.

Proposed methodology

Our approach

A methodology for designing intelligent tutors that can be implemented on commercially available frameworks.
It considers two main phases:

Knowledge Modeling describes how knowledge is represented and stored in the knowledge base (KB).

Conversation Flow deals with the lexicon used by the tutor and the order in which ideas are presented.

Proposed methodology

Our approach

A methodology for designing intelligent tutors that can be implemented on commercially available frameworks.
It considers two main phases:

Knowledge Modeling describes how knowledge is represented and stored in the knowledge base (KB).

Conversation Flow deals with the lexicon used by the tutor and the order in which ideas are presented.

Proposed methodology

Our approach

A methodology for designing intelligent tutors that can be implemented on commercially available frameworks.
It considers two main phases:

Knowledge Modeling describes how knowledge is represented and stored in the knowledge base (KB).

Conversation Flow deals with the lexicon used by the tutor and the order in which ideas are presented.

Formal definitions

Our approach

A chatbot can be described as a function f of the form $f : Q \to R$, mapping **queries** $q \in Q$ to **responses** $r \in R$.

Each query must be translated from natural language to a single given entry in the KB by identifying key concepts of the conversation.

The most notable concepts at play are **entities** and **intents**.

Formal definitions

Our approach

A chatbot can be described as a function f of the form $f : Q \to R$, mapping **queries** $q \in Q$ to **responses** $r \in R$.

Each query must be translated from natural language to a single given entry in the KB by identifying key concepts of the conversation.

The most notable concepts at play are **entities** and **intents**.

Formal definitions

Our approach

A chatbot can be described as a function f of the form $f : Q \to R$, mapping **queries** $q \in Q$ to **responses** $r \in R$.

Each query must be translated from natural language to a single given entry in the KB by identifying key concepts of the conversation.

The most notable concepts at play are **entities** and **intents**.

Entities

Our approach

An entity is an abstract object which holds relevance to the user.

It is usually **the subject** or an **object** in a conventional sentence. Who or what are we talking about?

The quick brown fox jumps over the lazy dog

Entities

Our approach

An entity is an abstract object which holds relevance to the user.

It is usually **the subject** or an **object** in a conventional sentence. Who or what are we talking about?

The quick brown fox jumps over the lazy dog

Entities

Our approach

An entity is an abstract object which holds relevance to the user.

It is usually **the subject** or an **object** in a conventional sentence. Who or what are we talking about?

The quick brown fox jumps over the lazy dog

Entities

Our approach

An entity is an abstract object which holds relevance to the user.

It is usually **the subject** or an **object** in a conventional sentence. Who or what are we talking about?

*The quick brown **fox** jumps over the lazy **dog***

Intents

Our approach

Intents are abstract representations of the user's intentions. It is something the user wants to do or know.

Not always present in a sentence.

- *Show me my agenda*
- *What's the time in New Zealand?*
- *Why do we snore?*
- *What's the difference between x64 and x86?*

Intents

Our approach

Intents are abstract representations of the user's intentions. It is something the user wants to do or know.

Not always present in a sentence.

- *Show me my agenda*
- *What's the time in New Zealand?*
- *Why do we snore?*
- *What's the difference between x64 and x86?*

Intents

Our approach

Intents are abstract representations of the user's intentions. It is something the user wants to do or know.

Not always present in a sentence.

- *Show me my agenda*
- *What's the time in New Zealand?*
- *Why do we snore?*
- *What's the difference between x64 and x86?*

Intents

Our approach

Intents are abstract representations of the user's intentions. It is something the user wants to do or know.

Not always present in a sentence.

- *Show me my agenda*
- *What's the time in New Zealand?*
- *Why do we snore?*
- *What's the difference between x64 and x86?*

Intents

Our approach

Intents are abstract representations of the user's intentions. It is something the user wants to do or know.

Not always present in a sentence.

- *Show me my agenda*
- *What's the time in New Zealand?*
- *Why do we snore?*
- *What's the difference between x64 and x86?*

Intents

Our approach

Intents are abstract representations of the user's intentions. It is something the user wants to do or know.

Not always present in a sentence.

- *Show me my agenda*
- *What's the time in New Zealand?*
- *Why do we snore?*
- *What's the difference between x64 and x86?*

Intents

Our approach

Intents are abstract representations of the user's intentions. It is something the user wants to do or know.

Not always present in a sentence.

- ***Show*** *me my agenda*
- *What's* ***the time*** *in New Zealand?*
- ***Why*** *do we snore?*
- *What's the* ***difference between*** *x64 and x86?*

Entities and Intents as functions

Our approach

We can (to some extent) rework the sentences and express them as **first-order logic functions**:

- `show(agenda)`
- `tell_time(NZ)`
- `reason(snoring)`
- `difference(x64, x86)`

$$g^n(t_1, t_2, \ldots, t_n)$$

which is an n-ary symbol in the set of Intents G, with terms t_i in the set of Entities T.

Entities and Intents as functions

Our approach

We can (to some extent) rework the sentences and express them as **first-order logic functions**:

- `show(agenda)`
- `tell_time(NZ)`
- `reason(snoring)`
- `difference(x64, x86)`

$$g^n(t_1, t_2, \ldots, t_n)$$

which is an n-ary symbol in the set of Intents G, with terms t_i in the set of Entities T.

Entities and Intents as functions

Our approach

We can (to some extent) rework the sentences and express them as **first-order logic functions**:

- `show(agenda)`
- `tell_time(NZ)`
- `reason(snoring)`
- `difference(x64, x86)`

$$g^n(t_1, t_2, \ldots, t_n)$$

which is an n-ary symbol in the set of Intents G, with terms t_i in the set of Entities T.

Entities and Intents as functions

Our approach

We can (to some extent) rework the sentences and express them as **first-order logic functions**:

- `show(agenda)`
- `tell_time(NZ)`
- `reason(snoring)`
- `difference(x64, x86)`

$$g^n(t_1, t_2, \ldots, t_n)$$

which is an n-ary symbol in the set of Intents G, with terms t_i in the set of Entities T.

Entities and Intents as functions

Our approach

We can (to some extent) rework the sentences and express them as **first-order logic functions**:

- `show(agenda)`
- `tell_time(NZ)`
- `reason(snoring)`
- `difference(x64, x86)`

$$g^n(t_1, t_2, \ldots, t_n)$$

which is an n-ary symbol in the set of Intents G, with terms t_i in the set of Entities T.

Entities and Intents as functions

Our approach

We can (to some extent) rework the sentences and express them as **first-order logic functions**:

- `show(agenda)`
- `tell_time(NZ)`
- `reason(snoring)`
- `difference(x64, x86)`

$$g^n(t_1, t_2, \ldots, t_n)$$

which is an n-ary symbol in the set of Intents G, with terms t_i in the set of Entities T.

Entities and Intents as functions

Our approach

We can (to some extent) rework the sentences and express them as **first-order logic functions**:

- `show(agenda)`
- `tell_time(NZ)`
- `reason(snoring)`
- `difference(x64, x86)`

$$g^n(t_1, t_2, \ldots, t_n)$$

which is an n-ary symbol in the set of Intents G, with terms t_i in the set of Entities T.

Knowledge storing

Our approach

1. Break down queries as knowledge units (FOL functions)
2. Team-up with course instructors and experts to generate responses
3. Store in a relevant data structure
   - Tree is the most common (Watson, DialogFlow)
   - Model each branch as a function (intent)
   - Model each tree node as a query (intent as function and entities as params)

Knowledge storing

Our approach

1. Break down queries as knowledge units (FOL functions)
2. Team-up with course instructors and experts to generate responses
3. Store in a relevant data structure
   - Tree is the most common (Watson, DialogFlow)
   - Model each branch as a function (intent)
   - Model each tree node as a query (intent as function and entities as params)

Knowledge storing

Our approach

1. Break down queries as knowledge units (FOL functions)
2. Team-up with course instructors and experts to generate responses
3. Store in a relevant data structure
   - Tree is the most common (Watson, DialogFlow)
   - Model each branch as a function (intent)
   - Model each tree node as a query (intent as function and entities as params)

Knowledge storing

Our approach

1. Break down queries as knowledge units (FOL functions)
2. Team-up with course instructors and experts to generate responses
3. Store in a relevant data structure
   - Tree is the most common (Watson, DialogFlow)
   - Model each branch as a function (intent)
   - Model each tree node as a query (intent as function and entities as params)

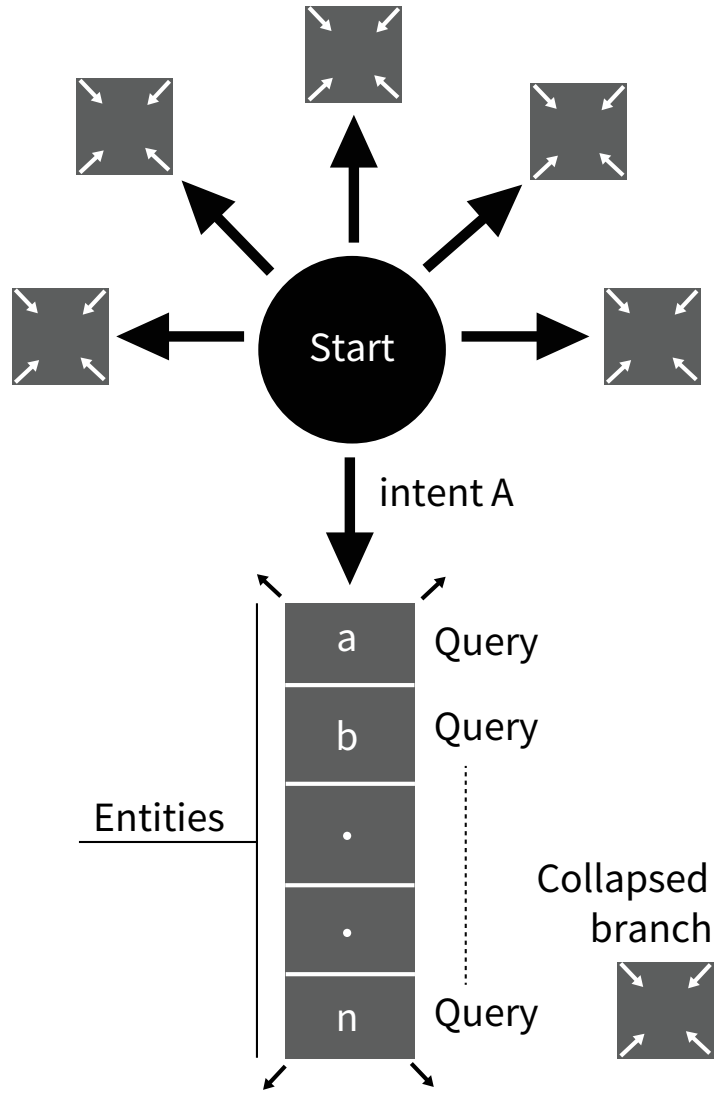
Knowledge storing

Our approach

1. Break down queries as knowledge units (FOL functions)
2. Team-up with course instructors and experts to generate responses
3. Store in a relevant data structure
   - Tree is the most common (Watson, DialogFlow)
   - Model each branch as a function (intent)
   - Model each tree node as a query (intent as function and entities as params)

Knowledge base built by grouping queries

Our approach



Conversation Flow

Our approach

Create a naming convention for intents and entities and then generate a glossary to easily identify queries.

ID	Intent	Entities
`def-N`	definition	Natural numbers
`ex-Q`	Example	Rational numbers
`subset-Q+R`	Subset	Rational numbers, Real numbers
`card-Qc`	Cardinality	Irrational numbers
`expl-change+unif`	Explanation	Uniform, Change

Conflicts

Our approach

The tree is really a *dictionary of arrays*, i.e. each branch is an array and is explored sequentially.

This could generate unwanted responses:

1. A user ask for `def(constant, velocity)`
2. The bot identifies the definition branch and explores it sequentially.
3. It detects two conditions: `constant` and `velocity`, However...
   - If *definition of velocity* exists, it could be triggered.
   - If *definition of constant* exists, it could also be triggered.

Conflicts

Our approach

The tree is really a *dictionary of arrays*, i.e. each branch is an array and is explored sequentially.

This could generate unwanted responses:

1. A user ask for `def(constant, velocity)`
2. The bot identifies the definition branch and explores it sequentially.
3. It detects two conditions: `constant` and `velocity`, However...
   - If *definition of velocity* exists, it could be triggered.
   - If *definition of constant* exists, it could also be triggered.

Conflicts

Our approach

The tree is really a *dictionary of arrays*, i.e. each branch is an array and is explored sequentially.

This could generate unwanted responses:

1. A user ask for `def(constant, velocity)`
2. The bot identifies the definition branch and explores it sequentially.
3. It detects two conditions: `constant` and `velocity`, However...
   - If *definition of velocity* exists, it could be triggered.
   - If *definition of constant* exists, it could also be triggered.

Conflicts

Our approach

The tree is really a *dictionary of arrays*, i.e. each branch is an array and is explored sequentially.

This could generate unwanted responses:

1. A user ask for `def(constant, velocity)`
2. The bot identifies the definition branch and explores it sequentially.
3. It detects two conditions: `constant` and `velocity`, However...
   - If *definition of velocity* exists, it could be triggered.
   - If *definition of constant* exists, it could also be triggered.

Conflicts

Our approach

The tree is really a *dictionary of arrays*, i.e. each branch is an array and is explored sequentially.

This could generate unwanted responses:

1. A user ask for `def(constant, velocity)`
2. The bot identifies the definition branch and explores it sequentially.
3. It detects two conditions: `constant` and `velocity`, However...
   - If *definition of velocity* exists, it could be triggered.
   - If *definition of constant* exists, it could also be triggered.

Conflicts

Our approach

The tree is really a *dictionary of arrays*, i.e. each branch is an array and is explored sequentially.

This could generate unwanted responses:

1. A user ask for `def(constant, velocity)`
2. The bot identifies the definition branch and explores it sequentially.
3. It detects two conditions: `constant` and `velocity`, However...
   - If *definition of velocity* exists, it could be triggered.
   - If *definition of constant* exists, it could also be triggered.

Conflicts

Our approach

The tree is really a *dictionary of arrays*, i.e. each branch is an array and is explored sequentially.

This could generate unwanted responses:

1. A user ask for `def(constant, velocity)`
2. The bot identifies the definition branch and explores it sequentially.
3. It detects two conditions: `constant` and `velocity`, However...
   - If *definition of velocity* exists, it could be triggered.
   - If *definition of constant* exists, it could also be triggered.

Conflict Resolution

Our approach

To prevent conflicts, re-order by placing more generic nodes (unary functions) at the bottom.

Alternative approach: additional level of branching + asking *Which of the following definitions of velocity are you interested in?*

Inefficient since there could be different topics on velocity (definition, example, notation, ...).

Conflict Resolution

Our approach

To prevent conflicts, re-order by placing more generic nodes (unary functions) at the bottom.

Alternative approach: additional level of branching + asking *Which of the following definitions of velocity are you interested in?*

Inefficient since there could be different topics on velocity (definition, example, notation, ...).

Conflict Resolution

Our approach

To prevent conflicts, re-order by placing more generic nodes (unary functions) at the bottom.

Alternative approach: additional level of branching + asking *Which of the following definitions of velocity are you interested in?*

Inefficient since there could be different topics on velocity (definition, example, notation, ...).

Implementation in our institution

Our approach

- F-1001: Physics introductory course
  - MOOC in Coursera: *Conceptos y Herramientas para la Física Universitaria*
- MA-1001: Mathematics introductory course
  - MOOC in Coursera: *El Cálculo (Modelo Lineal)*

Implemented in IBM's *Bluemix Conversation Service*:

- The tree structure is actually a `JSON` file (thus creation can be automated).
- Synonyms of entities are uploaded using `CSV` files.
- Training examples are uploaded using `CSV` files as well.

Many different chatbots have been created (most of them as FAQs consultants) using this methodology.

Implementation in our institution

Our approach

- F-1001: Physics introductory course
  - MOOC in Coursera: *Conceptos y Herramientas para la Física Universitaria*
- MA-1001: Mathematics introductory course
  - MOOC in Coursera: *El Cálculo (Modelo Lineal)*

Implemented in IBM's *Bluemix Conversation Service*:

- The tree structure is actually a `JSON` file (thus creation can be automated).
- Synonyms of entities are uploaded using `CSV` files.
- Training examples are uploaded using `CSV` files as well.

Many different chatbots have been created (most of them as FAQs consultants) using this methodology.

Implementation in our institution

Our approach

- F-1001: Physics introductory course
  - MOOC in Coursera: *Conceptos y Herramientas para la Física Universitaria*
- MA-1001: Mathematics introductory course
  - MOOC in Coursera: *El Cálculo (Modelo Lineal)*

Implemented in IBM's *Bluemix Conversation Service*:

- The tree structure is actually a `JSON` file (thus creation can be automated).
- Synonyms of entities are uploaded using `CSV` files.
- Training examples are uploaded using `CSV` files as well.

Many different chatbots have been created (most of them as FAQs consultants) using this methodology.

Implementation in our institution

Our approach

- F-1001: Physics introductory course
  - MOOC in Coursera: *Conceptos y Herramientas para la Física Universitaria*
- MA-1001: Mathematics introductory course
  - MOOC in Coursera: *El Cálculo (Modelo Lineal)*

Implemented in IBM's *Bluemix Conversation Service*:

- The tree structure is actually a `JSON` file (thus creation can be automated).
- Synonyms of entities are uploaded using `CSV` files.
- Training examples are uploaded using `CSV` files as well.

Many different chatbots have been created (most of them as FAQs consultants) using this methodology.

Implementation in our institution

Our approach

- F-1001: Physics introductory course
  - MOOC in Coursera: *Conceptos y Herramientas para la Física Universitaria*
- MA-1001: Mathematics introductory course
  - MOOC in Coursera: *El Cálculo (Modelo Lineal)*

Implemented in IBM's *Bluemix Conversation Service*:

- The tree structure is actually a `JSON` file (thus creation can be automated).
- Synonyms of entities are uploaded using `CSV` files.
- Training examples are uploaded using `CSV` files as well.

Many different chatbots have been created (most of them as FAQs consultants) using this methodology.

Implementation in our institution

Our approach

- F-1001: Physics introductory course
  - MOOC in Coursera: *Conceptos y Herramientas para la Física Universitaria*
- MA-1001: Mathematics introductory course
  - MOOC in Coursera: *El Cálculo (Modelo Lineal)*

Implemented in IBM's *Bluemix Conversation Service*:

- The tree structure is actually a `JSON` file (thus creation can be automated).
- Synonyms of entities are uploaded using `CSV` files.
- Training examples are uploaded using `CSV` files as well.

Many different chatbots have been created (most of them as FAQs consultants) using this methodology.

Thank you!

Any questions?

Xavier Sánchez Díaz
Research Group with Strategic Focus in Intelligent Systems
sax@itesm.mx