Yoda Easy Guide

David Cohen
ECE Department – Carnegie Mellon University

Introduction

YODA, Yet another Ontological Dialog Architecture is a sophisticated tool to allow developers to quickly build intelligent spoken dialog systems. This tutorial will introduce the major components and capabilities of Yoda by walking through the creation of a simple messaging calendar assistant. We envision this system as a smartphone app, which the user can use to message their friends and coordinate meetings.

Understand Your Dialog System's Desired Functionality

The YODA dialog manager is designed to build dialog systems which combine the functions of information retrieval (IR), information entry, and transactional dialog systems. An IR dialog system's main purpose is to answer questions about a database of objects - such as restaurants, movies, or events in a calendar. An information entry system allows the user to create descriptions of objects for entry in a database - such as scheduling events in a calendar. A transactional dialog system allows the system to take actions on behalf of the user, such as making reservations or sending emails.

The first step for a YODA system developer is to decide what databases it is intended to access and in what way, and what non-dialog actions their system can take (if it is transactional), and what types of objects it can talk about.

If your intended system does not fit into the categories described here, YODA is probably not an appropriate dialog manager for your project. We appreciate your feedback about why you feel this is the case, and we hope to support a broader set of dialog systems in the future. Please email David at david.cohen@sv.cmu.edu .

Set up of the YODA directory

Your dialog system will be built on top of the skeleton YODA dialog system. Once you've installed YODA, believe it or not, your dialog system already exists! (run bin/launch.sh).

Of course, this dialog system doesn't know very much - it knows what people and places are, but doesn't know about any particular person or place. The vast majority of your early work will be done in the yoda/ontology and yoda/interfaces directories. In the ontology folder, you will extend the dialog system's domain to allow it to talk about the objects you are interested in. In the interfaces directory, you will connect your dialog system to database resources it needs.

Build the Ontology

A YODA developer must then formalize the objects, actions and properties which are relevant to their domain by extending the YODA skeleton ontology with new classes, properties, values, and instances. Objects should be given appropriate slots, relations, and properties which correspond to the actual objects. There are several powerfull tools

available for editing OWL ontologies, but we recommend protege, and all our examples are shown in protege.

Figue 1 shows the specification of a person in the OWL skeleton ontology. Figure 2 shows the specification of a meeting, and of the domain action set-up-meeting. There are additional objects and actions that are defined for our meeting room reservation system, details of which can be found in the attached .owl file.

Define interfaces to External Databases

Classes in the ontology correspond to supporting databases.

There are two types of supporting databases, corresponding to two primary functions:

- **Long-term Memory Database:**Store objects from interactions in long-term memory to keep reasoning in working memory tractable while allowing for possible retrieval at a later date
- **Reference Database:** Store and index a large collection of reference objects (the standard use paradigm of a database)

By default, every class defined in the ontology has no supporting reference database, and an independent long-term memory database. If the developer wants to provide a reference database they may do so. The developer can decide weather or not the dialog system is allowed to insert to that database. If the dialog system is not allowed to insert, then a separate long-term memory database will be automatically created. Otherwise, the reference database will provide both long-term memory and reference functions.

Reference databases must be Sesame RDF databases. Many existing databases are in relational format and are accessed via SQL. These databases can be represented as RDF databases, as is described in A Survey of Current Approaches to Mapping of Relational Databases to RDF. YODA requires a map between slots and properties which are shared by the domain ontology and the database schema. YODA only allows a one-to-one mapping of properties and slots, but allows certain properties to exist only in the reference database.

Any database interface that does not have the default setting is defined by modifying the yoda/interfaces/DatabaseInterfaces.java file.

Define interfaces to Non-dialog actions

Similar to the database interface definitions, the developer can specify action interfaces to specify what information is needed to actually perform an action.

Build a lexicon for NLG

Initial linguistic information should be associated with each new class and instance. The two figures below show examples of the linguistic information associated with the objects specified above. The complete example is given in the attached owl file.

The program TestLexicalEntry.java generates sample sentences based on the lexicon provided by the developer. The developer can use this program to test that their lexicon generates sensible sentences and covers the full variety of things the developer expects to be said about the object. Call it from the command line using: java TestLexicalEntry.java -e [entryID]

Train SLU and speech modules with artificial data

To build an initial SLU component, YODA takes the generated lexicon and creates a large training set. It adds artificial noise to the training set for robustness, and extends the training set using wordnet synonyms. It then trains a state-of-the-art SLU component based on this artificial data.

To build an initial SLU component, run java GenerateInitialSLUComponent.java . Include -v in the command line arguments after the filename to see cross-validation results within the auto-generated corpus.

The training data set is retained in yoda/SLU/artificial_train.txt, and the domain-specific annotation scheme is described in yoda/SLU/annotation_scheme.txt . As real data is collected, the SLU component can be improved by incorporating annotated real-world examples.

To build an improved SLU component, put labeled real-world data in yoda/SLU/real_data.txt in the same format as artificial_rain.txt, then run: java GenerateImprovedSLUComponent.java . Include a -v in the arguments after the filename to see validation results and a comparison between the initial and the improved systems.

It will likely be the case that as real data is collected, the developer will discover that the domain and lexicon definitions should be extended to improve coverage of users actully say. We recommend that this be done by extending the ontology rather than simply training the improved model. The purpose of the improved model is to improve the component's likelihood model for different utterances, not to compensate for poor coverage.

To generate a language model based on the automatically generated sentences and the real data examples, run the script build_language_model.sh .

Build sensor interfaces

YODA dialog systems support situated interaction by defining sensor interfaces. Sensors can report the existance of new entities, and their properties and relations. The interfaces define a message protocol used to receive updates from the sensors.

Some of the information provided in the protocol includes:

- Expected framerate
- Expected consistency of the sensor readings over time
- Type of confidence markup (N-best, Bayes net, full CPT)
- Overlap interface (Define the overlap between this sensor and other sensors)
- Relevance interface (Choose a strategy for defining how relevance to the ongoing discourse should be tracked for objects in this sensor stream)
- Expected recall of objects (effects dialog strategy)
- Expected precision (of some slot?)

Sensor belief tracking in Yoda

YODA designers will explore belief-tracking and sensor fusion approaches that generalize to dynamic situated domains.

Long-term Memory and Discourse Relevance in YODA

YODA designers will explore different strategies for tracking objects' relevance to an ongoing discourse, and for using that relevance to move information between long-term and working memory. The different properties of different sensor types will be better suited for different relevance strategies.