An Ontology-Based Dialogue Management System for Virtual Personal Assistants

Michael Wessel, Girish Acharya, James Carpenter, and Min Yin

Abstract *Dialogue management (DM) is a difficult problem.* We present *OntoVPA*, an Ontology-Based *Dialogue Management System (DMS)* for *Virtual Personal Assistants (VPA's).* The features of OntoVPA are offered as potential solutions to core DM problems. We illustrate OntoVPA's solutions to these problems by means of a running VPA example domain. To the best of our knowledge, OntoVPA is the first commercially available, fully implemented DMS that employs ontologies, reasoning, and ontology-based rules for a) domain model representation and reasoning, b) dialogue representation and state tracking, and c) response generation. OntoVPA is a *declarative, knowledge-based system* which, consequently, can be customized to a new VPA domain by swapping in and out ontologies and rule bases, with *very little to no conventional programming required.* OntoVPA relies on its domain-independent (generic), but dialogue-specific upper-level ontologies and DM rules, which are implementing typical, re-occurring (and usually expensive to address) dialogue system core capabilities, such as anaphora (coreference) resolution, slot-filling, inquiring about missing slot values, and so on. We argue that ontologies and ontology-based rules provide a flexible and expressive framework for realizing DMS's for VPA's, with a potential to significantly reduce development time.

1 Introduction, Motivation, and Related Work

Motivation for OntoVPA *Dialogue management (DM)*, the core functionality of a *Dialogue Management System (DMS)*, is notoriously difficult, if not AI-complete; see [20] for a recent overview. Even in more restricted dialogue systems, difficult problems such as *anaphora (coreference) resolution* and *dialogue state tracking* may have to be handled by a non-trivial DMS. The importance and difficulty of the dialogue state tracking problem is also testified by the recently established series of *Dialog State Tracking Challenges* [19], aimed at catalyzing progress in this area.

To the best of our knowledge, state tracking and DM are still in its infancy in contemporary *commercial* VPA frameworks / platforms, with very little to no support offered by the frameworks. Commercial platforms usually offer some form of

Corresponding author: Michael Wessel
SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, USA, e-mail: firstname.lastname@sri.com

current user intent-triggered action-reaction ($\approx$ production) rules (e.g., in order to invoke a RESTful service in the Internet of Things). However, these rules are usually constrained to accessing the parameters ($\approx$ slot values) of the current user intent only, and hence cannot access (nor assess) the dialogue history or state of a rich domain model – their *context* is often limited to current intent and previous state. In contrast, our ontology-based DMS has access to the full dialogue history and domain model.

Consequently, in these simpler commercial systems, the bookkeeping required in order to support more sophisticated DM has to be done programmatically by the VPA application developer, and dialogue history and state have to be encoded in proprietary data structures, with little to no reuse across domains. This model-less, programmatic approach to DM is acceptable for simple *one-shot request-response* VPA's that do not need to sustain complex conversations (possibly involving multiple dialogue steps) for fulfilling a user request, but becomes tedious with increased development times and hence costs in the long run in more complex VPA domains that require elaborate workflows for solving domain-specific problems (e.g., booking a business trip includes booking a flight, a hotel, a rental car, and so on).

A system with deep domain knowledge which is capable to solve complex problems in cooperation with the user, such as Kasisto [9], is also called a *Virtual Personal Specialist (VPS)* [12]. Due to the complexity of the domain problems to be solved (e.g., bank transfers), dialogues will require multiple steps for workflow / intent completion, and both the user and the VPA should be able to steer and drive the dialogue in order to solve the problem cooperatively. Such systems will likely be *mixed initiative dialogue systems* [6], and *frame-based*, *information state-based* and *agent-based* DMS are a better fit than the less flexible *finite state machine-based* DMS. The challenges presented by conversational VPS's have shaped OntoVPA, our ontology-based DMS for VPA's.

Introducing OntoVPA OntoVPA is offered as a generic, reusable VPA platform for implementing conversational VPA's and VPS's that require deep domain knowledge, complex workflows, and flexible (not hard-coded) DM strategies. It promises to significantly reduce development times, due to its reusable and generic features (see Section 2).

OntoVPA employs ontologies for *dialogue and domain representation,* as well as *ontology-based rules* for *dialogue management, state tracking, and response computation.* OntoVPA's dialogue representation can be compared to the *blackboard* in *information state space-based* DMS's [20].

Ontology-Based Domain Model and Dialogue Representation For illustration, let us consider a *Restaurant Recommendation VPA* (see Section 2) that has suggested a specific restaurant of type *ItalianPizzaRestaurant* to the user in response to a $\mathit{FindRestaurantIntent}(\mathit{type} : \mathit{Restaurant})$. In subsequent dialogue steps, the user might refer to this previously discussed Restaurant with a phrase such as *... the pizza place ...* or *... the Italian restaurant ...* (e.g., "What is the rating of the pizza place?"). OntoVPA uses its domain knowledge to realize that the previously presented *ItalianPizzaRestaurant* is an *ItalianRestaurant* as well as a *PizzaPlace* –

consequently, these anaphora can be resolved with the help of a generic anaphora resolution rule. The rule considers the taxonomy of the ontology and is hence aware of hypernyms, hyponyms, and synonyms [7].

Such a system obviously requires some form of *dialog representation* in order to have a working memory of the dialogue, and it needs to be "reflexive" – in addition to what the user said, it also needs to remember its own utterances; here, the presented restaurant. The runtime, dynamic dialogue representation is instantiating the classes and relation types defined in the static dialogue ontology; this vocabulary is inspired by *speech act theory [14].* Ontology-based rules over these representations implement the *dynamics of the dialog.*

Use of Ontologies in OntoVPA We are using the W3C standards OWL 2 for the ontology language, and have implemented a custom ontology-based rule engine based on the SPARQL 1.1 RDF query language. Mature implementations for OWL 2 and SPARQL exists [16], and modeling workbenches (Protégé 5) are available. The use of standards facilitates customer acceptance, and "off the shelf" ontologies are only readily available in standard formats. *Ontologies facilitate the following:*

• **Domain Model Representation:** The classes (types), relationship types, and instances and relations of the VPA domain. Often, we wish to reuse, extend, and specialize well-known upper level ontologies such as Schema.org [18]. Following standard OWL 2 Description Logic (DL) terminology [17], the *classes (≈ concepts, types, . . .)* and *object and datatype properties (≈ properties, slots, attributes, relations, parameters, . . .)* constitute the *TBox (terminological box)* of the ontology, representing the *vocabulary* of the domain, whereas the actual instances of these classes and relationships between them are kept in the *ABox (assertional box)*.

• **Dialogue Ontology:** A TBox – its classes and properties are speech act theory-inspired [14]. This vocabulary is instantiated in the *dialogue ABox*, which is the actual *dialogue representation*. The TBox contains dialogue step classes, such as *UserIntent*s, which are special *UserRequest*s, which are special *DialogUserStep*s, and so on, see Section 3. As the domain classes, many dialogue steps have parameters (slots), which are defined in terms of object and datatype properties.

Ontology reasoning is applicable in a number of areas and offers great potentials:

• **Domain-Specific Reasoning:** A VPA with deep expertise in a domain (a VPS) requires extensive domain knowledge, and workflows and reasoning procedures are becoming more complex. Automatic reasoning and highly expressive ontology-based rule languages can solve complex application problems in a declarative way.

• **Reasoning to Compute VPA's Responses:** Ontology-based rules can be used to compute the actual utterances of the VPA.

• **Reasoning to Handle Polysemy & Ambiguity of Natural Language and Dialogue:** Ontologies provide a means to deal with the polysemy of natural language (NL). An ontology can structure the domain-specific vocabulary ("lexicon") in terms of semantic relations between them, such as hypernym / hyponym, syn-

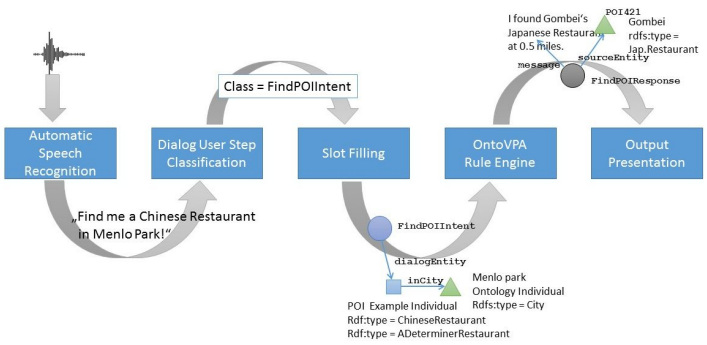

Fig. 1 OntoVPA's processing pipeline. *Automatic Speech Recognition (ASR)* turn the audio signal into text. The text is then classified as a subclass of the *DialogueUserStep* ontology class, i.e., a specific *UserIntent* such as *FindPointIntent*. The ontology also specifies parameters / slots for classes, and the *slot filler* "fills in" their parameters from a parse tree. The instantiated intent class (the *logical form*) refers to individuals, classes, and relations from the ontology. The *classifier and slot filler* act as a *semantic parser.* The instantiated intent is asserted into the dialogue ABox, and the *rule engine* is invoked to compute OntoVPA's response. The ontology is used by the classifier, slot-filler and rule engine. Only the rule engine uses the ontology-based SPARQL rules.

onym / antonyms, and so on. This knowledge can be exploited for a variety of context-specific NL interpretation and understanding tasks [7].

Basic Architecture of OntoVPA SRI's standard VPA processing pipeline is shown and explained in Fig. 1, also compare [6]. The focus of this paper is on the OntoVPA module. We will briefly discuss the role of the *classifier and slot filler*, too.

Related Work Since the days of the LUNAR system, ontologies have been used successfully in NL Question Answering systems, recently in HALO/AURA [10]. One of the few dialogue systems that uses ontologies at runtime for response generation is [1], but neither standard reasoning nor ontology-based queries / rules are used. Dynamic use of ontologies is considered in [2] and led to the OntoChef system [15], which is equipped with a sophisticated cooking ontology, but uses a non-ontology based DMS (Olympus/RavenClaw). Other case studies have focused on ontology modeling [11] and dialogue design based on task structures in OWL [3]. Frequently, ontologies are used for domain models. OWLSpeak [4, 5] is similar to our dialogue ontology in that it is capable to represent (speech act influenced) dialogue models as well as the state of the dialogue, but no ontology-based rules nor queries are used to compute the system's responses (it generates Voice-XML). A sophisticated Semantic Dialogue System for Radiologists is presented in [8], and it also relies on ontologies and SPARQL, but does not use these techniques for the implementation of the DM shell.

Outline of the Paper First, we illustrate OntoVPA and its DM core capabilities in a *Point Of Interest (POI) Finder* VPA domain by means of a typical dialogue. We then discuss the ontology in more detail, before we elaborate on how OntoVPA's DMS uses ontologies and ontology-based rules for realizing these core capabilities. We finish with a conclusion and outlook for future work.

No.	User Utterance (*DialogueUserStep*)	OntoVPA Response (*DialogueSystemStep*)
1	(none, OntoVPA initiates the dialogue by greeting the user upon sign in)	Hello, how can I help?
2	Is there a Chinese Restaurant in Menlo Park?	Sorry, I can't find any. Do you want me to find something similar?
3	Okay!	I found Gombei Japanese Restaurant in Menlo Park, at 0.34 miles.
4	Can you show it on the map?	I am assuming with 'it' you are referring to Gombei Japanese Restaurant. Now showing it on the map.
5	Is there a library close to the asian restaurant?	Yes, the Menlo Park library is close to Gombei Japanese Restaurant, at 0.3 miles.
6	How about a Pizzeria in Menlo Park?	I found Applewood Pizzeria in Menlo Park, at 0.2 miles.
7	In Palo Alto	I found Olive Garden in Palo Alto, at 1.4 miles.
8	Is there parking close to the pizza place?	Do you mean Olive Garden?
9	No	Apple Wood Pizzeria has a parking lot.

Fig. 2 An Example Dialogue with the OntoVPA-based "Point Of Interest Finder" VPA

2 Ontology-Based Dialogue Management Illustrated

Dialogue Management Core Capabilities What are the core capabilities required for conversational VPA's (also see [21, 7, 6])? In our experience, in addition to anaphora (coreference) resolution as already discussed, conversational VPA's typically require the following: realizing when a user request (also: *intent*) is fully specified and ready for execution (i.e., all *required parameters* are fully specified); inquiring about missing required parameters; interpreting *arbitrary user input* in the context of the current dialogue (what does "In Palo Alto?" mean in the current context?); canceling a currently open, but not yet fully executed sub-dialogue or sub-workflow; support for refining, generalizing, deleting or overwriting slot values of previously executed requests (intents), and re-executing them; and recognizing and disambiguating ambiguous input (does "Stanford" refer to the city, or the college?).

We will now illustrate some of these core capabilities by means of the example dialogue from Fig. 2 with a "Point of Interest Finder" VPA, and discuss how these are handled on a generic level in OntoVPA, cross-domain and "once and for all".

After an initial greeting from the system in **Step 1,** the user initiates the dialogue in **Step 2** by asking for a *Chinese Restaurant in Menlo Park.* The *domain ontology* contains a taxonomy of POI classes, such as *ChineseRestaurant*, which is a subclass (= kind) of *AsianRestaurant*, which is a kind of *Restaurant*, which is a kind of *POI*, and so on. In addition, there are classes such as *City*. The system also has a *domain data source*, which is an OWL (RDF) ABox of instances of POI classes (POI database for short). These POI's have their typical attributes (properties), i.e., name, address, geographic coordinates, and so on. Cities, such as *MenloPark*, are instances as well; a POI instance refers to the city in which it is located via the *inCity* object property (slot).

In Step 2 of Fig. 2, the user's request can be classified as a *FindPOIIntent*, a subclass of *UserIntent*. *DialogUserStep* classes are defined in the *dialogue ontology*; the dialogue ontology defines the vocabulary for the dialogue ABox, see Fig. 3. The user's utterances (often, instances of *UserRequest* or *UserIntent*) are normally created by the *semantic parser*, whereas OntoVPA's utterances (usually *SystemResponse*s) are created by OntoVPA's ontology-based rule engine. OntoVPA

keeps track of the current dialogue user step and current dialogue system step by annotating the corresponding instances in the dialogue ABox with so-called *control marker classes*, i.e. *CurrentDialogUserStep* and *CurrentDialogSystemStep*.

The *FindPOIIntent* is asserted into the dialogue ABox by the semantic parser, along with an instance of a *ChineseRestaurant* as filler for its *dialogueEntity* slot (object property). In addition, the parser realizes that "Menlo Park" is a name for the *MenloPark City* individual from the POI ABox, and fills it in for the *inCity* slot of the freshly created *ChineseRestaurant* instance, given that the range of the *inCity* property is *City*, and *MenloPark* is an instance of *City*. The freshly constructed *ChineseRestaurant* POI instance can be considered as a "query-by-example POI" for the *FindPOIIntent* query. We will discuss in Section 4 how a *generic, query-by-example semantic search* can be implemented with ontology-based rules.

OntoVPA now realizes that the *FindPOIIntent* is *completely specified* (i.e., all required parameters are specified), and hence is ready for execution – the semantic search over the POI database is performed. In this example, OntoVPA does not find a matching *ChineseRestaurant*s in *MenloPark*, and it takes the initiative by pro-actively asking the user whether the query should be *generalized.* Now, a *YesOrNoAnswer* is expected from the user. For both possible answers, OntoVPA has set up positive and negative *continuation requests;* the negative continuation is a *GreetingIntent*, whereas the positive continuation is a *FindPOIGeneralizedIntent*. The parameters required for the latter intent are copied over from the previous *FindPOIIntent*. Hence, depending on whether a *YesUserResponse* or *NoUserResponse* is received, the corresponding continuation is triggered automatically by OntoVPA. In **Step 3,** the *FindPOIGeneralizedIntent* intent is triggered based on the user's "Okay!" response, which is classified as a *YesUserResponse*.

For the *FindPOIGeneralizedIntent*, a *relaxed semantic matching condition* is implemented, where the structure of the taxonomy is exploited to compute a *semantic similarity measure* between the query-by-example POI, and the actual candidate source POI. A *JapaneseRestaurant* is more similar to a *ChineseRestaurant* than a *SteakHouse*, given that the former two have a common direct superclass *AsianRestaurant*, whereas *ChineseRestaurant* and *SteakHouse* do not.

In **Step 4,** "Can you show *it* on the map?", OntoVPA realizes that *it* refers to the most recently discussed POI. Like the already discussed *FindPOIIntent*, the *MapPOIIntent* has a *dialogueEntity* slot of range *POI*. The parser has created a "blank" *POI* instance that also instantiates the *ItDeterminerMixin* class – in OWL, individuals can instantiate multiple classes. The *ItDeterminerMixin* anaphora resolution rule now identifies the most recently discussed instance from the dialogue ABox that satisfies the given types, here: *POI*, and the most recent *POI* instance slot filler of any *UserIntent* or *SystemResponse* is identified as referent for *it.* Hence, the *sourceEntity* filler of the previous *FindGeneralizedPOISystemResponse* in the dialogue ABox is identified as the referent of the *it* anaphora. OntoVPA contains anaphora resolution rules for *it*, *the*, *a*, *his*, *her*, and so on.

Anaphora resolution involving the *TheDeterminerMixin* is illustrated in **Step 5** ("... *the* asian restaurant"). Realizing that the presented *JapaneseRestaurant* is also an instance of *AsianRestaurant* (a superclass), the anaphora can be resolved.

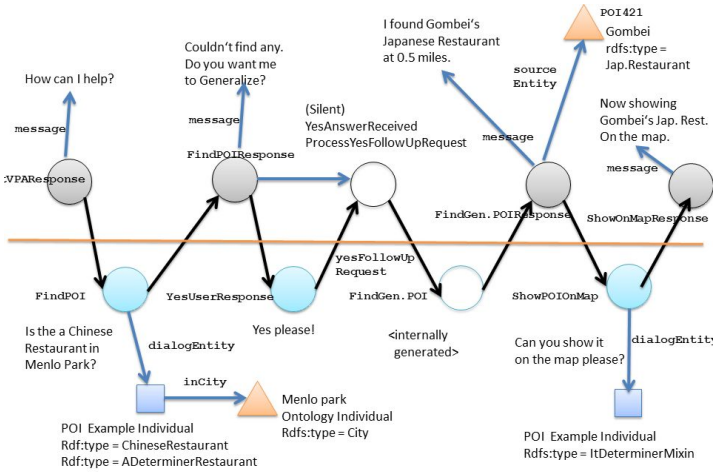

Fig. 3 Illustration of the Dialogue ABox. *UserDialogueSteps* are below the line, and *SystemDialogueSteps* above. The dialogue ABox after Step 4 is shown. Grey circles visualize OntoVPA's dialogue steps, mostly instances of *response classes*. Blue circles visualize user utterances, instances of *UserDialogueStep*. Blue shapes are created by the semantic parser; blue rectangles are instances of domain classes (e.g., *ChineseRestaurant*). Yellow triangle visualize ontology individuals from some data source (*MenloPark*). White circles are created programmatically via "follow up request processing" rules, and by the dialogue rule engine.

In order to demonstrate disambiguation, we are adding some more dialogue objects to the discourse, by requesting a *Pizzeria* in *MenloPark* in **Step 6**. In **Step 7** it is demonstrated how arbitrary input can be interpreted in the current context of the dialogue – based on the dialogue history, OntoVPA understands that the most likely user intent behind the ambiguous "In Palo Alto!" utterance is to *modify and re-execute the previous intent,* i.e., to look for a *Pizzeria* in *PaloAlto* instead of *MenloPark*. This introduces yet another *Pizzeria* instance.

Given that it is not self-evident what "In Palo Alto!" means without the context of the full dialogue, the semantic parser cannot instantiate a very specific *DialogueUserStep* or *UserIntent* class here (it does not have access to the dialogue representation). Instead, a generic high-level *ArbitraryUserInput* dialogue step with a *PaloAlto City* individual filler of the *inCity* slot is created, given that the proposition "in" maps to the *inCity* slot. A context-specific dialogue rule then processes the *ArbitaryUserInput* dialogue step – by looking at the *previousDialogueUserStep FindPOIIntent*, OntoVPA now suspects that the *inCity* slot value of the previous *FindPOIIntent* shall be overwritten with the given one (i.e., *MenloPark* be replaced with *PaloAlto*), and the so-modified *FindPOIIntent* be *re-executed.*

At **Step 8**, there are now two pizzerias in the dialogue ABox – since the *PizzaPlace* class is a synonym (= equivalent) class of the *Pizzeria* class, "the Pizza Place" is now ambiguous. One disambiguation strategy (out of several available) is to ask for clarification, as illustrated. OntoVPA also uses inference in **Step 9** in order to realize when the anaphora has been disambiguated, and the original

FindPOIIntent from Step 8 can be executed (notice that the exemplar *POI* instances also have an optional *nearBy* attribute, to which "close to" maps).

It should be noticed that none of these DM strategies are hard coded – things can be changed flexibly in terms of enabling, disabling, modifying or adding generic DM rules to OntoVPA's upper rule layer.

3 OntoVPA's Upper Ontology & Modeling in OWL

Upper Ontology Class Hierarchy OntoVPA's upper level ontology plays a key role for organizing and categorizing the different vocabularies into domain- and dialogue-specific parts, depending on role, categorizing *DialogueUnits* into certain speech acts, etc. The upper level ontology contains two main branches, the *upper-level dialogue ontology* branch and the *upper-level domain ontology* branch; the most important root classes are:

• **DialogueStep**: Root class of the dialogue ontology. Children of *DialogueStep* are: *DialogueUserStep*, *DialogueSystemStep*, *Request*, *Response*. Further down: *DialogueUserRequest*, *DialogueUserResponse*, *DialogueSystemRequest*, and *DialogueSystemResponse*. An important *DialogueUserRequest* subclass is *UserIntent*; all domain-specific intents such as *FindPOIIntent* specialize it. Arbitrary (highly dialogue context-dependent input) can be represented with *ArbitraryUserInput*, and special *Yes* and *No* response classes. Most intents have a corresponding *SystemResponse* class, e.g., *FindPOIResponse*.

• **DomainNotion**: Root class of the domain ontology. Only a few high-level notions such as *Entity*, *Event*, *TemporalThing*, and *SpatialThing*, are present. We also include Schema.org [18]. For example, the *AsianRestaurant* class will subclass *Schema.org/Restaurant*, and *ChineseRestaurant* will extend *AsianRestaurant*.

• **DialogueControlMarkers**: are used to control DM of the rule engine. We already mentioned *CurrentDialogueUserStep* and *CurrentDialogSystemStep*.

Upper Ontology Property Hierarchies In addition to the *class hierarchy,* the upper ontology also contains an *object property hierarchy* and a *datatype property hierarchy*. Like classes, OWL properties allows for multi-inheritance. The property hierarchies mirror the class hierarchy closely: there are root properties corresponding to the three main class-branches. Corresponding to the *DialogueStep* class, we have a *dialogueStepAttribute* object and *dialogueStepDatatypeAttribute* datatype property, with *DialogueStep* as corresponding domains. Next, we have *dialogueUserStepAttribute*, *dialogueSystemStepAttribute*, as well as the corresponding datatype properties. Important *dialogStepAttribute*s are *nextStep* and *previousStep*, *finalSystemResponse*; these relation types are used at runtime in the dialogue ABox; they can been seen as edges in Fig. 3. The *domain ontology property hierarchy* has root properties *domainAttribute* and *domainDatatypeAttribute* – for example, the *inCity* is subproperty of *entityAttribute*, which is a subproperty of *domainAttribute*. The third branch in the property hierarchies is given by the *controlAttribute* and

```
SubClassOf(
 POI
 ObjectIntersectionOf(
   SpatialEntity
   ObjectAllValuesFrom(
      inCity
      City)))
```

```
SubClassOf(
  FindPOIIntent
  ObjectIntersectionOf(
    ConcreteUserIntent
    ObjectSomeValuesFrom
      dialogueEntity
      POI)))
```

Fig. 4 The *POI* Domain Class and *FindPOIIntent* Dialogue Class in OWL 2 Functional Syntax

controlDatatypeAttribute properties. They are used to control the rule engine, and are often used in combination with control markers.

OWL Modeling of Domain Classes and Dialogue Steps Modeling properties of domain classes and intents in OWL seems to be straightforward – distinguishing between *required* and *optional* properties turns out to be challenging though, mainly due to the *Open World Assumption (OWA)* in OWL (and First-Order Logic). We argue that an *epistemic semantics* is needed in order to realize that an instance of a *UserIntent* class is *fully specified* and ready to be executed, i.e., *all of its required properties need to be explicitly specified* (either given by the user, or computed and filled-in based on context, by a rule). For example, the *FindPOIIntent* cannot be executed if the *required dialogueEntity* property is absent. Due to the OWA, declaring a property on a class by using an existential restriction of the form $\forall x$: $\mathit{FindPOIIntent}(x) \Rightarrow \mathit{dialogueEntity}(x,y)$ is not sufficient, as this is equivalent to $\forall x : \mathit{FindPOIIntent}(x) \Rightarrow \exists y : \mathit{dialogueEntity}(x,y)$. Given an "incomplete" dialogue ABox $\{\mathit{FindPOIIntent}(\mathit{userIntent1})\}$, the OWL reasoner will just assume that *some dialogueEntity* exists – this entity does not have to be *explicitly known*. Moreover, *POI* mentions the *inCity* property, but this should be an *optional* slot value on a *FindPOIIntent*.

So, how do we model *required and optional properties* in OWL, if these notions are problematic in terms of OWL semantics? From a logical point of view, every property is optional on an OWL class or instance, as long as it doesn't produce a logical inconsistency. For *required* properties (on *DialogueSteps*), we have adopted the convention of using the standard *existentially quantified* axioms, as just discussed: $\mathit{FindPOIIntent}(x) \Rightarrow \ldots \mathit{dialogueEntity}(x,y) \ldots \equiv \mathit{FindPOIIntent}(x) \Rightarrow \ldots (\exists y : \mathit{dialogueEntity}(x,y)) \ldots$. Such properties will always be interpreted under the stricter, "must be explicitly given" epistemic semantics on *DialogueStep*s, whereas properties which are declared using universal quantifiers such as $\forall x$: $\mathit{POI}(x) \Rightarrow \ldots \wedge (\forall y : \mathit{inCity}(x,y) \Rightarrow \mathit{City}(y)) \ldots$ are interpreted as *optional*.

4 Ontology-Based Rules for Dialogue Management & Workflows

Ontology-based rules are used to compute system responses, to implement DM strategies, domain workflows, and the majority of the domain-specific "application logic" in a declarative way. We adopted, adapted and extended the SPARQL 1.1 RDFs query language. SPARQL blends well with OWL – the employed Jena SPARQL engine [16] is aware of *inferred triples* in the OWL ABox caused by the

background axioms of the OWL TBox (ontology). SPARQL `construct` queries are used to dynamically augment the dialogue ABox (and potentially other ABox data sources) with conclusions. Since SPARQL is not a dialogue-specific rule language, we have created a custom rule engine for OntoVPA on top of a standard SPARQL query engine [16] which implements a discourse-specific rule interpreter and reasoner, including special rule application and conflict-resolution strategies in case more than one rule is applicable. The briefly mentioned *DialogueControlMarkers* play a crucial role in controlling and "advising" this rule engine. The rule engine is aware of the special semantics of the vocabulary in the dialogue ontology and implements a discourse-specific semantics for them.

A simple rule that responds to a *GreetingIntent* looks as follows in OntoVPA – it plays back the fixed *GreetingSystemResponse* defined in the ontology:

```
@
Hello-Toplevel
Reply to a greeting intent. Answer strings are defined in ontology.
1
CONSTRUCT { ?o vpa:assertedType vpa:CurrentDialogueSystemStepMarker }
WHERE
 { ?i vpa:assertedType vpa:CurrentUserIntentMarker .
   ?i vpa:assertedType vpa:GreetingIntent .
   ?i vpa:finalExpectedSystemResponse ?o }
```

A set of rules separated by @ is specified in a `.sparql` file. Each rule has a name; rules with a `-Toplevel` suffix act like "daemons" and are automatically checked for applicability and fired by the DMS. Non-daemon rules can be triggered by other rules, as *follow-up rules or continuations*. The third line provides control information, such as rule priorities and precedence information for defeasibility reasoning.

We like two mention *three features of SPARQL 1.1 that are essential* for OntoVPA. The **first essential feature** is the ability to `construct` new nodes and structure in the dialogue ABox by using "_:blank" nodes" in `construct`. These are "fresh" Skolems – for example we can construct a new *FinalResponse* instance `_:o` in the *GreetingIntent* rule as follows, instead of referring to the pre-constructed *finalExpectedSystemResponse* `?o`:

```
CONSTRUCT {
  _:o vpa:assertedType vpa:CurrentDialogueSystemStepMarker .
  _:o vpa:assertedType vpa:FinalResponse .
  _:o vpa:assertedType vpa:TextOutputModalityMixin .
  _:o vpa:message "Hi! How are you?" }
```

Frequently, fixed utterances are insufficient, and *templates* are used for answer generation. These can refer to bindings of query variable, and `rdfs:labels` from the ontology. The `xfn:concat` string concatenation SPARQL function is used to construct the answer strings. Hence, the **second essential feature** are SPARQL functions, similar to *procedural attachments*, which can also be defined by the user – arbitrary Java code can be executed if necessary, to interface with the outside world (e.g., SQL Database access).

Finally, the **third essential feature** of SPARQL is the ability to perform (a limited form of) *existential and universal second order quantification.* Consider the *FindPOIIntent* rule. This rule has to make sure that a candidate *sourcePOI* from the POI ABox fulfills all the requirements expressed in the exemplar *queryPOI*, i.e., it

must have (at least) all the properties of the by-example POI. This can be expressed as a second-order quantification over properties P: $\forall P : \forall z : P(queryPOI, z) \Rightarrow P(sourcePOI, z)$. Notice that *sourcePOI* can have more properties than required by the *queryPOI*. With some further refinements (we can restrict the quantification to *entityAttribute* subproperties), we can express this in SPARQL as follows:

```
?qentity vpa:assertedType ?qtype .
?sentity rdf:type ?qtype .
?sentity rdf:type vpa:SourceEntity .
FILTER NOT EXISTS {
   ?qentity ?par ?parVal .
   ?par rdfs:subPropertyOf vpa:entityAttribute .
   FILTER NOT EXISTS {
     ?sentity ?par ?parVal } }
```

This implements (a simplified) *generic semantic search* procedure, for all kinds of `vpa:SourceEntities`. Many of OntoVPA's generic DM capabilities (anaphora resolution etc.) are implemented succinctly and declaratively like this.

5 Conclusion & Outlook

We have presented the ontology-based DMS OntoVPA, and illustrated its underlying techniques. During the last 3 years, OntoVPA has been successfully deployed to 4 different SRI customers, ranging from domains as different as Beauty Consultant, Shopping Assistant, Car Conversational System, to an Augmented Reality Tutoring & Mentoring System. One of these VPA's uses Japanese language (internally, English is being used). OntoVPA supports *multi-modal* input and output – different input channels are represented using dedicated *inputModality* slots on the asserted *DialogUserStep*s, and likewise, dedicated output modality properties are used on the computed *SystemResponse* instances – `construct` is able to create arbitrarily complex (nested, cyclical, ...) output structures.

In the above projects, we have observed a significant reduction in development time and costs, compared to previous VPA projects @ SRI that did not use OntoVPA. The exact numbers and evaluation are subject to future research.

Basing OntoVPA and its dialogue representation on formal, explicit, standardized symbolic representations has the benefit of transparency and reusability – the dialogue history can be introspected, inspected and visualized (for example, using OWL visualizers); shared, persisted, resumed, etc.

OntoVPA is very flexible and expressive – the DM strategies employed by the system are easy to change if necessary. The system is highly expressive and can also "emulate" different DMS paradigms. For example, it is straightforward to implement a finite state machine-based DMS in OntoVPA. Moreover, the system is *reflexive* and can also be used on the *meta level* – instead of encoding the DM strategies as high-order rules as discussed previously, it is possible to encode these strategies on an instance level in the ABox and create higher-order *meta interpreter DM rules* that interpret the instance-level encoded DM strategies. We hence suspect that OntoVPA subsumes most of the existing DMS architectures on the market.

In order to evaluate OntoVPA's performance, we are musing about participating in a future *Dialog State Tracking Challenge* [19]. To conclude, we like to mention that OntoVPA can be licensed from SRI International.

References

1. E. Wantroba, R. Romero, *A Method for designing Dialogue Systems by using Ontologies*. Standardized Knowledge Representation and Ontologies for Robotics and Automation, 18th Sep. 2014, Chigago, USA.
2. J. Pardal, *Dynamic Use of Ontologies in Dialogue Systems*. Proceedings of the NAACL-HLT 2007 Doctoral Consortium, Association for Computational Linguistics, April 2007.
3. G. Liu, *A Task Ontology Model for Domain Independent Dialogue Management.* Electronic Theses and Dissertations, University of Windsor, Paper 5412, 2012.
4. T. Heinroth, D. Denich, A. Schmitt, W. Minker, *Efficient Spoken Dialogue Domain Representation and Interpretation*. Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010).
5. S. Ultes, H. Dikme, W. Minker, *Dialogue Management for User-Centered Adaptive Dialogue.* In "Situated Dialog in Speech-Based Human-Computer Interaction", 2016, Springer International Publishing, pp 51–61.
6. C. Lee, S. Jung, K. Kim, D. Lee, and G.G. Lee, *Recent Approaches to Dialog Management for Spoken Dialog Systems.* Journal of Computing Science and Engineering (JCSE), 4(1):1–22 (2010), April 2010.
7. D. Milward, M. Beveridge, *Ontology-Based Dialogue Systems.* Proceedings of the 3rd Workshop on Knowledge and Reasoning in Practical Dialogue Systems (IJCAI 2003), August 2003.
8. D. Sonntag, M. Huber, M. Möller, A. Ndiaye, S. Zillner, and A. Cavallaro, *Design and Implementation of a Semantic Dialogue System for Radiologists.* In Semantic Web: Standards, Tools and Ontologies, Kimberly A. Haffner (Eds), 2010 Nova Science Publishers.
9. Kasisto and KAI - http://kasisto.com, and http://kasisto.com/kai/. Accessed 2/10/2017.
10. D. Gunning, V. Chaudhri, P. Clark, K. Barker, S. Chaw, M. Greaves, B. Grosof, A. Leung, D. McDonald, S. Mishra, J. Pacheco, B. Porter, A. Spaulding, D. Tecuci, and J. Tien, *Project Halo Update – Progress Toward Digital Aristotle.* AI Magazine, October 2010, AAAI Press.
11. V. Chaudhri, A. Cheyer, R. Guili, B. Jarrold, K. Myers, and J. Niekarsz, *A Case Study in Engineering a Knowledge Base for an Intelligent Personal Assistant.* In Proceedings of the 5th International Conference on Semantic Desktop and Social Semantic Collaboration, 2006.
12. W. Mark, *The Rise of Virtual Specialists.* https://www.sri.com/blog/deep-knowledge-and-rise-virtual-specialists. Accessed 2/10/2017.
13. J.-F. Yeh, C.-H. Wu, M.-J. Chen, *Ontology-Based Speech Act Identification in a Bilingual Dialog System Using Partial Pattern Trees*, Journal of the American Society for Information Science and Technology, Volume 59 Number 5, Wiley Subscription Services, pp 684–694, 2008.
14. J.R. Searle, *Speech Acts. An Essay in the Philosophy of Language.* Cambridge University Press, Jan 2, 1969.
15. J. Pardal, *Starting to Cook a Coaching Dialogue System in the Olympus Framework.* In Proceedings of the Paralinguistic Information and its Integration in Spoken Dialogue Systems Workshop, pp 255–267, Springer, 2011.
16. Apache Jena - https://jena.apache.org/. Accessed 2/10/2017.
17. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.
18. Schema.org - http://schema.org/. RDFs exports http://schema.rdfs.org/. Accessed 2/10/2017.
19. J. Williams, A. Raux, D. Ramach, and A Black, *The Dialog State Tracking Challenge.* In Proceedings of the 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL), 2013.
20. P. Lison, *Structured Probabilistic Modelling for Dialogue Management.* Diss. University of Oslo, 2013.
21. C. Vertan, W. v. Hahn, *Project "Spoken Dialogue Systems"*. https://nats-www.informatik.uni-hamburg.de/pub/DIALSYS/VeranstaltungsMaterial/DialogueManagement.pdf, Seminar Slides. Accessed 4/13/2017.