

# An Ontology-Based Dialogue Management Framework for Virtual Personal Assistants in Common Lisp

Michael Wessel  
michael.wessel@sri.com  
SRI International  
Menlo Park, California, USA

## ABSTRACT

We present a new approach to *dialogue specification for Virtual Personal Assistants (VPAs)* based on so-called *dialogue workflow graphs*. Our approach relies on Semantic Web technology (OWL), implemented in Common Lisp with the help of the Racer reasoner. Our new *dialogue specification language (DSL)* is a set of Common Lisp macros, a Domain Specific Language, which facilitates customer participation in the modeling process. The resulting dialogue models are also very concise. The DSL is a new modeling layer on top of our *ontology-based Dialogue Management (DM) framework OntoVPA*. We explain the rationale and benefits behind the new language, and support our claims with concrete reduced Level-of-Effort (LOE) numbers from two recent OntoVPA projects.

## CCS CONCEPTS

• **Software and its engineering** → **Model-driven software engineering**; **Very high level languages**; **Domain specific languages**; **State based definitions**; **Application specific development environments**; **Software design tradeoffs**; **Rapid application development**; • **Information systems** → **Web Ontology Language (OWL)**; **Ontologies**; **Business intelligence**; • **Human-centered computing** → **Participatory design**; • **Computing methodologies** → **Discourse, dialogue and pragmatics**; **Description logics**; **Ontology engineering**.

## KEYWORDS

Knowledge-Based Dialogue Modeling, Knowledge-Based Workflow Modeling, Ontology-Based Dialogue Management, Semantic Natural Language Processing, Domain Specific Modeling Languages, Semantic Web, OWL, SPARQL, Common Lisp, Macros

### ACM Reference Format:

Michael Wessel. 2022. An Ontology-Based Dialogue Management Framework for Virtual Personal Assistants in Common Lisp. In *Proceedings of the 15th European Lisp Symposium (ELS'22)*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.5281/zenodo.6335631>

## 1 INTRODUCTION & MOTIVATION

In 2021, *Virtual Personal Assistants (VPAs)* have become commonplace on our smartphones and smart speakers.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ELS'22, March 21–22, 2022, Porto, Portugal

© 2022 Copyright held by the owner/author(s).

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.

<https://doi.org/10.5281/zenodo.6335631>

The VPA landscape ranges from simple “one-shot request-response” systems to control appliances and lights in an IoT home automation network, over more sophisticated *task-oriented virtual specialists* à la KASISTO [14] that help with complex tasks such as online banking, to freeform conversational chatbots that aim at passing the Turing Test.

With growing demands and user’s expectations regarding the “intelligence” and “human-likeness” of VPAs, there is clearly a need for DM systems that go beyond the simple “one-shot request-response” model.

A recent Gartner blogpost reads as follows [8]:

Gartner predicts that by 2025, 50 % of knowledge workers will use a virtual assistant on a daily basis, up from 2 % in 2019.

In 2022, users expect systems that are capable of supporting a) elements of freeform chat with contextual memory (e.g., learn about the user and current situation), b) are capable of supporting complex workflows and multiple turns, and c) can combine both in a non-rigid, non-linear, naturally flowing conversation that doesn’t feel like a telephone hotline support system. We have embraced and addressed some of these challenges in our *OntoVPA framework* [33, 35].

Most VPAs rely on one central component – the Dialogue Manager. The Dialogue Manager is typically concerned with *Dialogue Management (DM)*, which encompasses *Dialogue State Tracking* [37], and it also implements the *Dialogue Policy*: the computation of the system response (“system turn”) based on the current state of the world and dialogue / discourse (including the previous and current “user turns”).

A commonly used *conceptual* DM model is the *Dialogue Flow Graph*. We are referring to this graph as a *conceptual* model because the actual dialogue model representation might be different from a graph (e.g., in case of a rule-based manager). Still, the flow graph is a useful conceptual notion of high utility for developers and customers, given its comprehensibility. Unsurprisingly, various commercial solutions for “visual drag & drop” chatbot construction exist.

In the following, we prefer the term *Dialogue Workflow Graph (DWG)* because it emphasizes that this graph may also contain elements (“nodes and edges”) of *workflow management*, i.e., elements that describe the workflow actions that need to be carried out behind the scenes of the dialogue.

In this paper, we are presenting a novel DWG Specification Language in Common Lisp, *Dialogue Specification Language (DSL)* for short, and demonstrate its potential for significantly reducing the VPA modeling effort. Our DSL is also a *Domain Specific Language* in the conventional sense. It is built on top of OntoVPA [33, 35].

OntoVPA is a declarative, knowledge-based dialogue manager, in which new VPA domains can be implemented with very little to no conventional programming effort. OntoVPA is highly expressive and comes with built-in solutions to standard DM problems such as state tracking, anaphora resolution, contextual intent slot-filling, intent management, stack-based sub-dialogue management, etc. The rules that implement these common DM capabilities are *generic* and *cross-domain*. They are typically expensive to implement from scratch. As we will illustrate in Section 2.2, by relying on expressive ontology reasoning and forms of higher-order logic quantifications, these rules can *succinctly cover large regions in the DM problem space*, hence greatly reducing the modeling effort, yet achieving generality and hence cross-domain reusability.

Unfortunately, a much bigger *Level-of-Effort (LOE)* is required to model the dialogue workflows by means of rules. These are obviously VPA domain-specific and hence cannot be transferred cross-domain easily. From our experience in developing 5 VPAs with OntoVPA, this LOE can easily account for more than 70 % of the overall LOE.

Our DSL has shown great potential to reduce and simplify this LOE. It also has the benefit of being more intuitive and much less technically involved: unlike plain OntoVPA, developers no longer need to use OWL and SPARQL exclusively. *Visual* workflow dialogue graphs can be generated automatically from the textual DSL specifications, which facilitates customer participation, transparency, and comprehensibility. *The DSL hence has the potential to significantly reduce the modeling effort, thus widening the dialogue workflow modeling bottleneck.*

The remainder of this paper is structured as follows. We first describe the OntoVPA framework, providing the basis for this work. The essence of ontology-based DM is illustrated by means of typical DM problems in the "Restaurant Recommendation" domain [13]. OntoVPA relies on Common Lisp and Racer [11] for the OWL DM modeling and reasoning at *development time* - we hence describe the modeling environment and OntoVPA compiler. At *runtime*, the compiled (target) ontology and a set of ontology-based SPARQL rules are being executed by the JENA reasoner [6, 19] to implement the runtime dynamics. We then subsequently describe three different workflow representation options, corresponding to abstraction layers. The lowest layer (Level 1) corresponds to plain OntoVPA dialogue workflow modeling, and the final (Level 3) abstraction layer corresponds to the new DSL. Next we give a concrete illustration of the DSL, utilizing Common Lisp macros. We then quantitatively demonstrate the benefits of the DSL in terms of reduced modeling LOE. We conclude with a discussion of related work and a summary.

## 2 ONTOLOGY-BASED DIALOGUE MANAGEMENT

The original OntoVPA is described elsewhere in full detail [33, 35]. It realizes many of the ideas first described by Milward and Beveridge [21, 22], using modern Semantic Web languages and frameworks.

Let us first summarize OntoVPA's distinguished features and architecture so that we can describe the DSL improvements "on top" of the OntoVPA model.

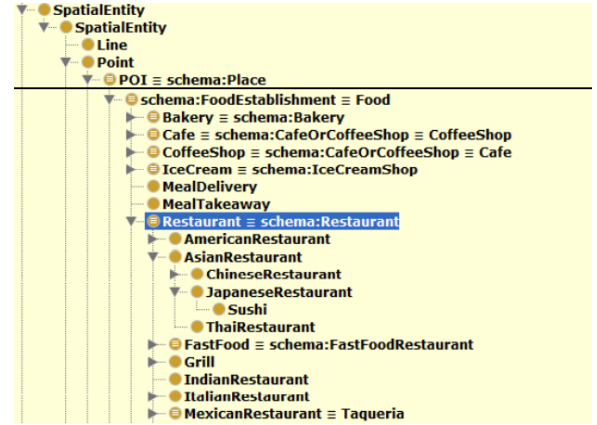


Figure 1: Domain-Specific Ontology

### 2.1 The Original OntoVPA Model & Architecture

OntoVPA employs the *Web Ontology Language (OWL)* for knowledge representation and reasoning [27]. Additionally, it uses *ontology-based SPARQL* [28] rules for DM which are executed in a custom VPA-specific rule engine built on top of the JENA reasoner [6, 19]. In what follows, we assume basic familiarity with OWL notions such as class, instance, and property, and likewise for SPARQL [3].

These SPARQL rules are SPARQL queries extended with some extra-annotations: rules have IDs, priorities for conflict resolution, can pass control to other rules, etc. This results in a custom, DM-specific ontology-based highly expressive rule language and engine: the *OntoVPA rule engine*.

An OntoVPA model has the following main components:

- An OWL ontology for *background and domain knowledge* – e.g., for the Restaurant Recommendation VPA it will have classes (concepts), relations (OWL object and datatype properties) and instances (individuals) for representing (types of) restaurants, cities, different cuisines, an actual database of restaurant instances, etc. Frequently, we extent and reuse Schema.org [24]; see Figure 1.
- An OWL ontology of *classes and relations for dialogue / discourse representation* – this includes classes for user and system turns, intents and their slot values, and system response classes. *Speech act theory* [25] provides us with a coarse but useful upper ontology, for Requests, Response, etc.; also compare [5] for a similar upper-level ontology. A portion of OntoVPA's upper-level ontology is shown in Figure 2. At runtime, these classes and relations are instantiated in an *OWL ABox*, which is a set of class instances and relationships, representing the actual dialogue, i.e., the history of user and system dialogue turns. For example, a restaurant recommendation VPA will have a user intent class *FindRestaurantIntent*, and a corresponding system response class *SuggestRestaurantResponse*, and relations (object properties) for "slots" such as cuisine and location.
- A layer of *Generic Dialogue Management Rules* – these rules implement core DM capabilities, like slot-value filling, anaphora resolution, context-based disambiguation, details of

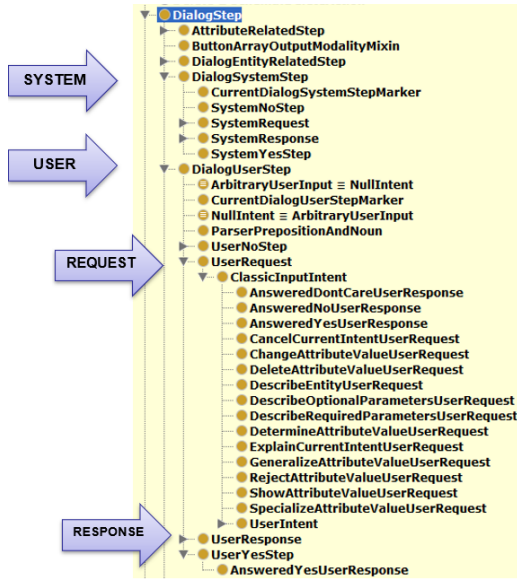


Figure 2: OntoVPA Upper Ontology - Subclasses of *DialogStep*

turn taking, etc. Not every domain requires all these capabilities; they can be disabled individually, but also refined as needed.

- VPA domain-specific rules that implement the dialogue workflow and intent processing. In a Restaurant Recommendation VPA, there will be rules that trigger if certain intents are instantiated, e.g., a *RecommendRestaurant* rule triggers if the current intent is a *FindRestaurantIntent*. The rule checks for the presence of the requested location and cuisine, then employs ontology-based query answering to retrieve matching restaurant instances and presents the result.

We heavily rely on *upper-level ontologies and inheritance*, and on generic and reusable behavior specified in the generic DM rules. The combination of ontology reasoning with expressive, succinct higher-order ontology-based rules is the defining feature of OntoVPA.

## 2.2 Ontology-Based Dialogue Management Example

Let us illustrate the *interplay of ontology reasoning, generic DM rules, and domain-specific rules that implement the dialogue workflow* by means of the following dialogue in the Restaurant Recommendation Domain:

**User** I am looking for a restaurant!

**VPA** In what city?

**User** In Palo Alto.

**VPA** How about McDonalds?

**User** Chinese please.

**VPA** Got it – Su Hong on 4256 El Camino Real?

The first utterance can clearly be classified as a *FindRestaurantIntent* by the statistical intent classifier – let us assume this intent class has a required slot location (of range *City*), and an optional slot cuisine (of range *Cuisine*). This intent is instantiated in the dialogue representation ABox, but without any slot values.

A generic DM rule now inspects the current intent and the definition of its associated OWL intent class and determines that a required slot value (the location slot) is missing. To check for the presence or absence of slot values, this generic SPARQL rule uses *existential quantification over slots / predicates*; consequently, we consider this a *higher-order rule*. It hence triggers a follow-up question to the user, who subsequently answered "Palo Alto".

The string "PaloAlto" is automatically mapped to the corresponding OWL *City* instance of the same name. Next, another generic DM rule determines that the *PaloAlto* individual is substitutable for the inquired location slot value of the previous intent – the user has answered the question. Hence, *PaloAlto* is filled in as a slot value of the previous *FindRestaurantIntent*, and the intent is now marked *completely specified* by another DM rule. This subsequently triggers the domain-specific workflow *RecommendRestaurant* rule, which proceeds as already described. It retrieves and presents a single restaurant to the user, who subsequently requests "Chinese" instead of the presented option.

Since no intent was recognized from the word "Chinese" in isolation, the system relies again on the dialogue history for sense making / understanding in context. The ontology-based parser has mapped "Chinese" to the class *ChineseCuisine*, which is a subclass of *Cuisine*. From the dialogue history it determines that *ChineseCuisine* is a potential slot-filler for the cuisine slot on the previous *FindRestaurantIntent*, and it is hence augmented with the optional slot filler. Given that the intent has changed, it is then re-executed by a rule that detects the change and which then triggers re-execution of the *RecommendRestaurant* rule.

Many of OntoVPA's generic DM rules are specified in a similar flavor. Central expressive means of these higher-order generic SPARQL rules are the ability to

- introspect the OWL class definitions (of intents),
- traverse and assess the full dialogue history,
- perform ontology reasoning (e.g., sub-class and sub-property inferences) in the rule itself,
- existentially and universally quantify "in a second-order fashion" over arbitrary classes and slots (properties) rather than having to codify individual rules for individual classes and slots,
- create and update arbitrarily nested and complex structured graph representations (in the ABox); SPARQL allows for the creation of Skolem instances in the right-hand side of rules, and of arbitrary atoms / structure,
- specify defeasibility and priorities on rules.

These DM rules are supplied in an upper level, reusable rule layer applicable to any domain – they provide generic DM capabilities.

In addition, OntoVPA modelers typically still must write domain specific rules like the *RecommendRestaurant* rule, which is the highest LOE activity that we are seeking to replace with the DSL here.

## 2.3 Rationale for Using a Lisp Language Layer

The OntoVPA compiler is written in Common Lisp with Racer [11]. Given that the runtime engine of OntoVPA is implemented in Java with JENA [6, 19], why have Racer and Common Lisp in the loop at all? We could have just used Protégé [23] for OWL modeling (and

```
(implies FindRestaurantIntent
  (and SchemaOrgSearchIntent
    (some location City)
    (all cuisine Cuisine)))

(implies City Location)
(instance PaloAlto City)
(implies ChineseCuisine Cuisine)
```

**Figure 3: Source Ontology Fragment in Racer Syntax**

SPARQL rules are written by hand anyway). We are using the Racer & Common Lisp-empowered tool chain for the following reasons.

Common Lisp provides an ideal framework for implementing the DSL. Its macros, and most importantly, the ability to flexibly adjust, extent, and change the DSL for the current VPA project under development makes Common Lisp ideal for resource-constrained research-oriented prototyping projects in which high agility and flexibility is required. In fact, there never was a single DSL, and it was never “designed” – the DSL evolved dynamically over and in different projects, sometimes even incorporating customer-specific terminology to increase model comprehensibility for the customer and hence customer participation. It is likely that these projects would have failed or would have required substantially more development time if a more formal, less agile, “plan and implement” waterfall modeling approach had been taken. After all, implementing a fresh DSL in more traditional DSL environments usually requires specifications of meta-models (MOFs), grammars, parsers, etc., greatly adding to the development time and ultimately, project development costs.

Racer is implemented in Common Lisp, providing convenient and mature Lisp-based OWL modeling and reasoning. OWL modeling in Lisp has many advantages, is concise and readable, and Common Lisp macros can reduce the amount of boiler-plate modeling representations to a minimum. Racer’s built-in OWL parsers enable use to reuse existing OWL and RDFs ontologies (such as Schema.org [24]) by importing these, combining, extending and blending them with our dialogue model, post-processing the unified combined model, and then exporting it in a standard OWL syntax (OWL RDF XML, OWL Functional) utilizing one of Racer’s OWL renderers. The source ontology definition of the just discussed FindRestaurantIntent is illustrated in Figure 3.

Racer is used to perform (static) inferences about the domain and dialogue model at development time for quality assurance and ultimately, trust; e.g., for identifying inconsistencies, redundancies, and implied relationships between classes and relations. Racer is mature and offers reliable, high performance for non-trivial medium-sized ontologies.

Racer then transforms (“compiles”) the development time (source) ontology into a runtime (target) ontology that is loaded into JENA together with the (hand-authored) ontology-based SPARQL rules.

The main rationale for the compilation process is to *avoid most (if not all) potentially expensive JENA OWL inferences at runtime*, as VPAs must be very reactive. We achieve this by compiling the source ontology into a light-weight version of itself, and by making most relevant inferences explicit at compile time (anticipating them) via *inference materialization*. At runtime, OWL inference for

```
<RequiredParameterSpec rdf:about="#param21">
  <functional rdf:datatype="XSD#string">true</functional>
  <range rdf:resource="#City"/>
  <parameter rdf:resource="#location"/>
  <assertedType rdf:resource="#RequiredParameterSpec"/>
  <assertedType rdf:resource="#ParameterSpec"/>
</RequiredParameterSpec>

<OptionalParameterSpec rdf:about="#param66">
  <functional rdf:datatype="XSD#string">true</functional>
  <range rdf:resource="#Cuisine"/>
  <parameter rdf:resource="#cuisine"/>
  <assertedType rdf:resource="#OptionalParameterSpec"/>
  <assertedType rdf:resource="#ParameterSpec"/>
</OptionalParameterSpec>

<owl:Thing rdf:about="#FindRestaurantIntent">
  <representativeConcept
    rdf:resource="#FindRestaurantIntent"/>
  <finalExpectedSystemResponse
    rdf:resource="#RecommendRestaurantResponse"/>
  <requiredParameterSpec rdf:resource="#param21"/>
  <optionalParameterSpec rdf:resource="#param66"/>
  <assertedType rdf:resource="#DialogUserStep"/>
  <assertedType rdf:resource="#UserRequest"/>
  <assertedType rdf:resource="#UserIntent"/>
  ...
  <assertedType rdf:resource="#SchemaOrgSearchIntent"/>
</owl:Thing>
```

**Figure 4: Target Ontology Fragment in OWL RDF/XML**

JENA is hence then mostly reduced to slot-value lookups, or simple taxonomic query answering, which is extremely fast.

As illustrated in Figure 4 using the target ontology definition of the FindRestaurantIntent, this inference materialization process has computed the deductive closure of the taxonomy via the assertedType property, and has also pre-computed the required and optional slot values of intents. In general (but not in the FindRestaurantIntent), intent superclasses will inherit required property values to their subclasses, and these will need to be specified in a dialogue with the user. The ranges of property values (e.g., City) might as well enforce yet additional necessary property values (e.g., City might have required name and geolocation properties). The materialization process pre-computes required and optional properties as well as their characteristics (e.g., their ranges and cardinalities). Note the optional/requiredParameterSpec properties and corresponding specification instances in Figure 4. The deductive closure is a finite subset of the full deductive closure of the ontology (which might be infinite). In general, we *avoid disjunctions in the ontology*. Disjunctions can be dealt with *at runtime* in the dialogue via SPARQL rules, but don’t need to be present in the OWL ontology.

The OntoVPA compiler also facilitates the generation of visual representation; we are using DOT and Graphviz [7, 9].

### 3 MODELING DIALOGUE WORKFLOWS

Our underlying conceptual model is the

- *Dialogue Workflow Graph (DWG)*. Like a (labeled) transition system [36], this graph represents the space of possible dialogues and workflows, and hence the policy of the system.

This is a development time (static) representation. At runtime, this conceptual graph is interpreted and operationalized by the Dialogue

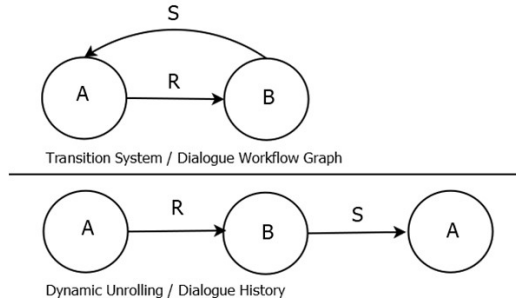


**Figure 5: Illustration Transition System & Unrolling**

```

CONSTRUCT {
  _:s vpa:assertedType vpa:B .
  _:s vpa:message "Transitioned to B!" .
  _:s vpa:assertedType vpa:CurrentSystemStepMarker .
  ?i vpa:computedSystemResponse _:s }
WHERE
{ ?i vpa:assertedType vpa:CurrentUserStepMarker .
  ?i vpa:assertedType vpa:R .
  ?i vpa:previousUserStep ?p .
  ?p vpa:computedSystemResponse ?pr .
  ?pr vpa:assertedType vpa:A }

```

**Figure 6: Simple SPARQL Transition Rule**

Manager to compute the system answers / turns. This results in the dynamic

- *Dialogue (Discourse) History*, which is an OWL ABox in OntoVPA. OWL instances are representing user and system turns (nodes), which are asserted and computed, with their causal, temporal, and thematic attributes and relationships.

The dialogue history can be considered an unrolling of the conceptual DWG, analog to the unrolling of a transition system into a trace of dynamic system behavior, see Figure 5.

In this abstract example, A might represent a *ConfirmRestaurantLocation* state, R a corresponding user utterance ("In Palo Alto!"), and B a *SuggestRestaurant* system response.

The dialogue history employs a notion of *current user turn* and *current (last) system turn*. User turns are simply asserted in the dialogue history (and made current) by the intent classifier and ontology-based slot-filler, whereas the system turns are computed by the Dialogue Manager.

In the following, we describe three *concrete* representation options for the conceptual DWG, corresponding to *layers of representations* in OntoVPA. Level 1 is the original OntoVPA representation, Level 3 the new DSL, and Level 2 an intermediate representation.

### 3.1 Level 1 – Rule-Based Workflow Graph Modeling

It is straightforward to operationalize a conceptual DWG, or transition system, via rules:

For each labeled edge  $A \xrightarrow{R} B$  in the DWG, create a corresponding rule that, given current state  $A$ , creates a new successor node of state  $B$  if condition  $R$  holds.

A corresponding OntoVPA SPARQL rule is shown in Figure 6. If, in the WHERE antecedent, the current user step satisfies  $R$  and the previous system response was of type  $A$ , then the CONSTRUCT consequent of the SPARQL rule creates a new node of type  $B$  via the  $_:s$  Skolem constant constructor. The freshly constructed node will be asserted as current system step into the dialogue history, and the asserted message property will cause a system utterance. Note that  $A$ ,  $B$ , and  $R$ , are OWL ontology classes and properties, and that the SPARQL engine is aware of the deductive closure of the OWL ontology (i.e., its inferences).

The type and number of *conditions* ( $R$  in the example) can be arbitrarily complex and may involve full OWL / Description Logic reasoning over the current dialogue, workflow completion, and world state.

There is no limit on what can be asserted with a CONSTRUCT consequent of a SPARQL rule at dialogue runtime – not only can we update the dialogue history, but also other parts of the ABox. For example, personal information learned about the user could be stored on the user's profile information in the ABox.

Let us discuss two obvious drawbacks of the just discussed rule-based modeling style are: a) the large number of rules required, and b) the amount of boilerplate code needed (e.g., each SPARQL rule must determine the *CurrentUserStep* in the history, construct the follow-up node, mark it as *CurrentSystemStep*, etc.)

Regarding a), the number of rules is roughly given by

$$\#nodes * average\_node\_out\_degree$$

This number is *quickly in the three-figure range* for non-trivial dialogue models, see Table 1 in Section 4.

*Problem b)* can be alleviated by using our *SPARQL macros*. However, the "graph as a set of rules" representation is also unwieldy from a modeling perspective, and a direct graph representation would be preferable. Moreover, SPARQL modeling is intellectually demanding and requires expert knowledge.

### 3.2 Level 2 – Graph-Based Workflow Graph Modeling

Our first step of remediation is hence to represent the workflow graph *directly as a graph* in the OWL ABox, in terms of *instances (nodes) and relationships (edges)*.

Unlike a set of SPARQL rules, an ABox graph representation is not an executable specification – hence, a *workflow graph interpreter* is required that operationalizes the graph. We chose to implement the interpreter (in a meta-circular way) *itself* in terms of OntoVPA's SPARQL rules, instead of extending the JENA-based rule engine.

The ABox-based, explicit and direct graph representation has some modeling benefits – SPARQL domain rules no longer need to be authored by the domain developer. The interpreter rules are part of the OntoVPA framework already and usually do not need to be altered, just like the generic DM rules already discussed. Moreover, DWGs can now be inspected and, in principle, also be edited visually, by means of OWL tools such as Protégé [23] and RacerPorter [32, 34].

One drawback of this representation though is that it requires *additional modeling vocabulary*, e.g., classes and relationships for representing logical conditions and control structure, as we can

```
(node "In node n1!"
  n1
  (:condition A)
  (:transition (R n2)))

(node "Transitioned to node n2"
  n2
  (:condition B)
  (:transition (S n1)))
```

Figure 7: DSL-Based Transition Specification

no longer rely on SPARQL specifications. The representation is *considerably less succinct* – a graph of hundred nodes might require a few thousand ABox assertions (see Table 1 in Section 4 for concrete numbers). The direct representation hence suffers from a *boilerplate representation problem* and complicates modeling due to a lack of tool support for our extra- and control-vocabulary.

Hence, a final abstraction layer added on top, *Level 3*. This layer provides the high-level DSL which is *translated (compiled) into the Level 2 representation* just discussed. We can consider Level 2 as semantic virtual machine code produced from Level 3 DSL high-level specifications, whereas Level 1 provides the generic implementation of the virtual machine that implements the DM / VPA instruction set in OntoVPA.

### 3.3 Level 3 – DSL-Based Workflow Graph Modeling

In our DSL, a workflow graph node is described together with its outgoing transitions and conditions. In Figure 7, the abstract transition system example from Figure 5 is specified.

Transitions are activated by *edge conditions*, and in addition, there is a *notion of active and disabled nodes*, based on the truth values of their corresponding *node conditions*. The interpreter will never transition into a disabled node.

A variety of *condition types* exist. These condition types are specified via different clauses using the node macro. A more complete illustration of the DSL is given in Figures 8 & 9, using the MEDIC VPA domain (see Section 4). Here, the `:next` clause specifies an unconditional transition to the successor node, and the `:utterance-domain` clauses correspond to the discussed `:transition` clauses.

Node activation and transition conditions can be specified based on:

- the *current intent and its slot values*. This information is usually asserted by a statistical intent classifier and the ontology-based slot filler. However, training statistical intent classifiers can be costly (in terms of training time and data requirements), and we hence support building VPAs without them, by offering ontology-based parsing of raw textual input, which can be used to implement intent classification or information extraction of slot-values or similar “text snippets” from the raw input text.
- parsing raw textual user input (e.g., from ASR) via *ontology-based grammar expressions*. The domain ontology then also plays the role of a *lexicon or thesaurus*, providing domain terms, and its *hyponyms, hypernyms, synonyms, and antonyms*. We support *regular expressions over ontology terms* – for example, the expression (Neg Amp1 PosDesc) is satisfied by the utterance “not very good”, or “not so well”, and might be used to transition to some other node. Sometimes, simple

```
(node "Massive Hemorrhage Control."
  node_mhc1
  (:immediate-transition t)
  (:next node_mhc2))

(node "Where is the bleeding?"
  node_mhc2
  (:utterance-domain
    ("Limb" node_mhc_limb)
    ("HeadOrNeck" node_mhc_head_or_neck)
    ("BodyPart" node_mhc_head_or_neck))
  (:extract-and-store
    ("BodyPart" "currentUser" "hemorrhageLocation"))))

(node "Ok, ${currentUser.hemorrhageLocation} hemorrhage."
  node_mhc_head_or_neck
  (:immediate-transition t)
  (:next node_mhc_check_if_clamp_applicable
    node_mhc_dont_apply_clamp))

(node "Can the wound edges be easily re-approximated?"
  node_mhc_check_if_clamp_applicable
  (:utterance-domain
    ("!vpa:AnsweredYesUserResponse" node_mhc_apply_clamp)
    ("!vpa:AnsweredYesUserResponse" node_mhc_apply_dressings)
    ("!vpa:AnsweredNoUserResponse" node_mhc_apply_dressings))
  (:negated-condition
    ("currentUser"
      ("hemorrhageLocation")
      ("nearTo" (:individual "Eye")))))
```

Figure 8: DSL Illustration from the MEDIC VPA

ontology-based keyword spotting is sufficient as well. We usually accept a hyponym for the specified ontology term in the expression, but this is not a strict requirement.

- complex (negated) *logical conditions that are evaluated over the ABox*, having access to the dialogue history and (ABox) domain model. Logical conditions are specified as *path expressions of OWL properties and classes*, starting from either an ABox individual (that then has to satisfy the expression), or from a class name (then there needs to be some instance that satisfies the expression). For example, in Figure 8 & 9, two possible `:next` transitions are specified on the `node_mhc_head_or_neck` node (yellow node in Figure 9): `node_mhc_check_if_clamp_applicable` and `node_mhc_dont_apply_clamp`. The former is only *active* (and a transition into it possible) if its `:negated-condition` evaluates to false, i.e., the `hemorrhage-Location` must not be `nearTo` the `Eye` (the haemostatic clamp cannot be applied to stop the bleeding in this case).

It is not always desirable or feasible to anticipate all potential dialogue transitions in terms of outgoing conditioned edges on nodes at development time. To allow a more naturally flowing and less linear dialogue, we allow transitions into different dialogue graph regions, e.g., corresponding to different conversation topics, *without requiring explicit outgoing edges on nodes leading into these regions*.

To facilitate these transitions into different regions, we employ the notion of a *trigger* for a node. If the trigger on a node is satisfied, control can transition into the trigger-activated node *in a non-local way, i.e., no edge needs to be transitioned to arrive at it*.

Additional annotations are specified on the node-level, determining aspects of control and thus direct the interpreter

- whether the node is a *start or end node* of a “topic”, i.e., a *sub-graph* that initiates a dialogue / conversation about a specific topic,

- if it *can be triggered*, and hence gain control, if a certain *trigger condition* is satisfied (i.e., an intent or special event has been recognized and been asserted as current),
- whether the node allows to *relinquish control* to another node via trigger-based / non-local transitions,
- whether the dialogue should return and resume at that point after the node lost control in a non-local way (i.e., after the triggered "sub-dialogue" has completed),
- if, in case of a topic end node, the control should *return* to the previously active node, or simply continue;
- if the node is *modal*, i.e., waiting for input before continuing / transitioning to the next node, or if the transition is :immediate; immediate nodes are useful for breaking up the workflow into smaller chunks, among other things.

Finally, the last group two groups of node annotations allow us to a) generate output and b), to update the ABox.

For a), each node can be annotated with a multitude of messages using the :message field. We support *template-based NLG* – rather than using variables for the template "holes", we are yet again using ontology-based path expressions that "peek into the ABox" to fill the template holes; see Figure 9.

Regarding b), we can assert and retract arbitrary ABox assertions. An example from a OntoVPA Medical Decision Support System is shown in Figure 8: the medic is asked for the location of a bleeding. Any answer subsumed by the BodyPart class is then extracted from the utterance and stored as a slot value on the currentUser individual's hemorrhageLocation slot, using the :extract-and-store clause. Depending on the specific BodyPart, the system then transitions into different branches (for Limb, HeadOrNeck).

#### 4 QUANTIFYING THE DSL BENEFITS

Table 1 shows the *number of nodes*, *number of rules that did not have to be modeled*, *number of ABox graph assertion generated by the DSL compiler*, *the average number of Rules per Nodes (RpN)*, and *the number of Assertion per Node (ApN)* for 2 domains, ChatPal and MEDIC:

| Domain  | #Nodes | #Rules saved | #Assertions | RpN | ApN |
|---------|--------|--------------|-------------|-----|-----|
| ChatPal | 109    | 291          | 2520        | 2.7 | 23  |
| MEDIC   | 29     | 26           | 374         | 0.9 | 11  |

**Table 1: Quantifying the DSL Benefits**

**ChatPal:** A virtual personal companion for the elderly that aims to overcome loneliness and social isolation. ChatPal flexibly engages the senior in templated conversations about specific topics, such as hobbies, the senior's life history, family, relationships, etc., and it learns about the users' interests, hobbies, and relationships and is able to leverage the learned knowledge in subsequent sessions. The generated visual DWG can be found online; follow the URL under [30].

**MEDIC:** A simple Medical Decision Support System prototype. It implements the workflows from a standard medical procedure handbook. Using the DSL, its workflow dialogue graph was developed in  $\approx 20$  hours only. Again, the visual DWG can be found online; follow the URL under [31].

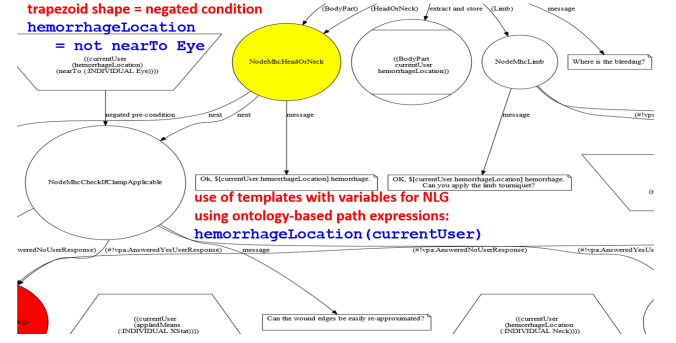


**Figure 9: Crop of Generated MEDIC Visual DWG**

From our experience, each Level 1 OntoVPA rule modeled by an OntoVPA expert accounts for  $\approx 1.5$  hours LOE. In contrast, MEDIC was modeled using DSL with 0.68 hours LOE per DSL node; modeled on Level 1, the corresponding 26 rules would have required  $26 * 1.5 = 39$  hours – a reduction by 48.7 %.

For ChatPal, the situation is even more significant:  $1.5 * 291 = 436.5$  vs.  $109 * 0.68 = 74.12$  hours – a reduction by 83 %.

#### 5 RELATED WORK

Knowledge bases and ontologies have been used since the early days of the LUNAR system [38] for Natural Language Understanding, Question Answering and Dialogue Systems, and, more recently, in HALO [10]. Frequently, (OWL) ontologies are being used by the Dialogue Manager for ontology-driven question answering relying on domain ontologies and external knowledge sources [2], [29].

Other case studies focused on ontology modeling and dialogue design based on task structures represented in OWL [4]. OntoDM [1] uses OWL ontologies for domain representation and partially for the dialogue history and NLU tasks such as anaphora resolution [22], but unlike OntoVPA it relies on special purpose algorithms that are informed by the ontologies, and is thus not fully declarative.

OWLSpeak [12] is based on Information State Theory [17], but the state is not implemented in OWL at runtime; hence, no ontology-based rules are being used. VONDA [15] is similar to OntoVPA in that it employs OWL and uses a proprietary RDFs/OWL-inference aware "reactive rules" language for policy specification, whereas OntoVPA relies on extended SPARQL over an ABox.

The Convergence system [20] is a hybrid system that uses OWL2 for most aspects of dialogue and domain representation, SPARQL endpoints for question answering, and defeasibility rules for context-aware reasoning [18]. The rules support disambiguation and conflict resolution over the dialogue discourse. Stoyanchev and Johnston [26] implement the Information State Approach [17], using Knowledge Graphs, also drawing inspiration from OntoVPA [33, 35].

An early approach to a generic, visual dialogue flow specification language is given by Kölzer [16]: a compiler translates visual dialogue flow graphs into Prolog knowledge bases.

The dozens of contemporary commercial visual flow builders for chat bot development (Google's Dialogflow, bots, Visual Chatbot Builder, etc.) usually require some form of programming for customization once the boundaries of the narrowly defined standard domains are reached (e.g., pre-defined models are supplied that

don't generalize well). We believe that OntoVPA's DSL provides a more general and more productive modeling environment.

## 6 CONCLUSION

We have made steps towards improving dialogue modeling efficiency and customer participation for ontology-based VPAs with our new dialogue modeling language, DSL. It corresponds to a new abstraction layer on top of OntoVPA, comes with a less technical DWG modeling syntax, a compiler and visualizer, and was successfully applied in 2 projects, reducing dialogue modeling effort significantly (by an estimated 49 % and 83 %, respectively).

Our solution is not based on visual programming or modeling; we believe that the throughput, modeling efficacy and efficiency of textual modeling languages is higher and leads to more general solutions.

## ACKNOWLEDGMENTS

I am grateful for the fruitful collaborations, support, ideas, and leadership provided by the following (ex-) SRI colleagues over the last 6 years: Girish Acharya, David Berends, Edgar Kalns, Min Yin, James Carpenter, Andreas Kathol, and Theodore Camus.

We would also like to thank Karen Myers and the anonymous reviewers for feedback and suggestions that significantly helped to improve the paper.

The work on MEDIC was supported by the US Army Medical Research and Materiel Command under Contract No. W81XWH-19-C-0096. The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision unless so designated by other documentation.

The development of OntoVPA and ChatPal was supported and funded by SRI.

## REFERENCES

- [1] Duygu Altınok. An Ontology-Based Dialogue Management System for Banking and Finance Dialogue Systems, 2018.
- [2] G. Amores, G. Pérez, P. Manchón, F. Gómez, and J. González. Integrating OWL Ontologies with a Dialogue Manager. *Proces. del Leng. Natural*, 37, 2006. URL [http://grupo.us.es/julietta/publications/2006/pdf/Integrating\\_OWL.pdf](http://grupo.us.es/julietta/publications/2006/pdf/Integrating_OWL.pdf).
- [3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook – Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [4] Vinay K. Chaudhri, Adam Cheyer, Richard Guili, Bill Jarrold, Karen L. Myers, and John Niekraz. A case study in engineering a knowledge base for an intelligent personal assistant. In *In the Proc. of the 2006 Semantic Desktop Workshop*, 2006.
- [5] Enrique Fernández-Rodicio, Á. González, F. Alonso-Martín, Marcos Maroto-Gómez, and M. Salichs. Modelling Multimodal Dialogues for Social Robots Using Communicative Acts. *Sensors (Basel, Switzerland)*, 20, 2020.
- [6] Apache Software Foundation. Apache Jena Apache Jena – A free and open source Java framework for building Semantic Web and Linked Data applications., Accessed: 9-14-2021. URL <https://jena.apache.org/>.
- [7] Emden R. Gansner and Stephen C. North. An Open Graph Visualization System and its Applications to Software Engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11):1203–1233, 2000.
- [8] Anthony J. Bradley / Gartner. Brace Yourself for an Explosion of Virtual Assistants, 8-10-2020. URL [https://blogs.gartner.com/anthony\\_bradley/2020/08/10/brace-yourself-for-an-explosion-of-virtual-assistants/](https://blogs.gartner.com/anthony_bradley/2020/08/10/brace-yourself-for-an-explosion-of-virtual-assistants/).
- [9] Graphviz. Graphviz & DOT Project Page, Accessed: 9-14-2021. URL <https://graphviz.org/>.
- [10] David Gunning, Vinay K. Chaudhri, Peter E. Clark, Ken Barker, Shaw-Yi Chaw, Mark Greaves, Benjamin Grosf, Alice Leung, David D. McDonald, Sunil Mishra, John Pacheco, Bruce Porter, Aaron Spaulding, Dan Tecuci, and Jing Tien. Project Halo Update—Progress Toward Digital Aristotle. *AI Magazine*, 31(3):33–58, Jul. 2010. doi: 10.1609/aimag.v31i3.2302. URL <https://ojs.aaai.org/index.php/aimagazine/article/view/2302>.
- [11] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The RacerPro Knowledge Representation and Reasoning System. *Semantic Web Journal*, 3(3): 267–277, 2012.
- [12] Tobias Heinroth, Dan Denich, and Alexander Schmitt. OwlSpeak - Adaptive Spoken Dialogue within Intelligent Environments. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 666–671, 2010. doi: 10.1109/PERCOMW.2010.5470518.
- [13] Matthew Henderson, Blaise Thomson, and Jason Williams. The Second Dialog State Tracking Challenge. In *In Proceedings of the SIGdial 2014 Conference*, 2014.
- [14] Kasisto. Kasisto company website, Accessed: 9-14-2021. URL <https://kasisto.com/kai/>.
- [15] Bernd Kiefer, Anna Welker, and Christophe Biwer. VONDA: A Framework for Ontology-Based Dialogue Management, 2019.
- [16] A. Kölzer. Universal dialogue specification for conversational systems. *Electronic Transactions Artificial Intelligence (ETAI)*, 3:33–52, 1999.
- [17] S. Larsson and D. Traum. Information State and Dialogue Management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6:323–340, 2000.
- [18] Thanassis Mavropoulos, Georgios Meditskos, Spyridon Symeonidis, Eleni Kamateri, Maria Rousi, Dimitris Tzimikas, Lefteris Papageorgiou, Christos Eleftheriadis, George Adamopoulos, Stefanos Vrochidis, and Ioannis Kompatsiaris. A Context-Aware Conversational Agent in the Rehabilitation Domain. *Future Internet*, 11(11), 2019. ISSN 1999-5903. doi: 10.3390/fi11110231. URL <https://www.mdpi.com/1999-5903/11/11/231>.
- [19] B. McBride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.
- [20] Georgios Meditskos, Efstratios Kontopoulos, Stefanos Vrochidis, and Ioannis Kompatsiaris. Convergence: Ontology-driven conversational awareness and context understanding in multimodal dialogue systems. *Expert Systems*, 37(1), 2020. doi: 10.1111/exsy.12378. URL <https://doi.org/10.1111/exsy.12378>.
- [21] David Milward. Ontologies and the structure of dialogue. In *Proceedings of the 8th Workshop on the Semantics and Pragmatics of Dialogue (Catalog)*, pages 69–77, 2004.
- [22] David Milward and Martin Beveridge. Ontology-based Dialogue Systems. In *Proceedings of the 3rd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Acapulco, Mexico*, pages 9–18, 2003. URL <http://www.ida.liu.se/labs/nlplab/ijcai-ws-03/papers/milward.pdf>.
- [23] Mark A. Musen. The Protégé Project: A Look Back and a Look Forward. *AI Matters*, 1(4):4–12, 2015. doi: 10.1145/2757001.2757003. URL <https://doi.org/10.1145/2757001.2757003>.
- [24] Schema.org. Schema.org Project Page, Accessed: 9-14-2021. URL <https://schema.org/>.
- [25] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969. doi: 10.1017/CBO9781139173438.
- [26] Svetlana Stoyanchev and Michael Johnston. Knowledge-Graph Driven Information State Approach to Dialog. In *AAAI Workshops*, 2018.
- [27] W3C. OWL 2 Web Ontology Language Document Overview (Second Edition), Accessed: 9-14-2021. URL <https://www.w3.org/TR/owl-overview/>.
- [28] W3C. SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013, Accessed: 9-14-2021. URL <https://www.w3.org/TR/sparql11-query/>.
- [29] E. Wantroba and R. Romero. A Method for Designing Dialogue Systems by Using Ontologies. In *Standardized Knowledge Representation and Ontologies for Robotics and Automation*, volume 58, pages 89–123, September 2014.
- [30] M. Wessel. Auto-Generated Visual Dialogue Workflow Graph for ChatPal, Accessed: 9-14-2021. URL <https://www.michael-wessel.info/downloads/chatpal.pdf>.
- [31] M. Wessel. Auto-Generated Visual Dialogue Workflow Graph for Medic, Accessed: 9-14-2021. URL <https://www.michael-wessel.info/downloads/medic.pdf>.
- [32] M. Wessel. RacerPorter Project Page, Accessed: 9-14-2021. URL <https://github.com/lambda-damikel/RacerPorter>.
- [33] M. Wessel, G. Acharya, J. Carpenter, and M. Yin. OntoVPA - An Ontology-Based Dialogue Management System for Virtual Personal Assistants. In M.; Devillers L.; Mariani J. Eskenazi, editor, *Advanced Social Interaction with Agents*, volume 510 of *Lecture Notes in Electrical Engineering*, pages 219–233. Springer, 2019.
- [34] Michael Wessel and Ralf Möller. Design Principles and Realization Techniques for User Friendly, Interactive, and Scalable Ontology Browsing and Inspection Tools. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, volume 258, 01 2007.
- [35] Michael Wessel, Girish Acharya, James Carpenter, and Min Yin. OntoVPA - An Ontology-Based Dialogue Management System for Virtual Personal Assistants. In *International Workshop on Spoken Dialogue Systems, IWSDS 2017*, 2017.
- [36] Wikipedia. Transition system, Accessed: 9-14-2021. URL [https://en.wikipedia.org/wiki/Transition\\_system](https://en.wikipedia.org/wiki/Transition_system).
- [37] Jason Williams, Antoine Raux, and Matthew Henderson. The Dialog State Tracking Challenge Series: A Review. *Dialogue & Discourse*, 7:4–33, 04 2016. doi: 10.5087/dad.2016.301.
- [38] William Woods. *Lunar Rocks in Natural English: Explorations in Natural Language Question Answering*, volume 5, pages 521–569. 01 1977.