Video Retrieval Using Natural Language: From Parse Tree to Database

Behrang Qasemizadeh, Ian O'Neill, Philip Hanna, Darryl Stewart

ECIT, The Institute of Electronics, Communications and Information Technology,
Queen's University Belfast, Northern Ireland Science Park, Queen's Road, Queen's Island,
Belfast, BT3 9DT, Northern Ireland.

{i.oneill,p.hanna,dw.stewart}@qub.ac.uk, b.qasemizadeh@ecit.qub.ac.uk

Abstract

In this paper we focus on the process of matching parsed natural language input with tagged video content in a database. The work is being undertaken as part of a new spoken dialogue system, ISIS-NL, a subcomponent of the EPSRC-funded ISIS project (EP/E028640/1), which aims to further safety on public transport through use of a multimodal sensor network. Here we describe the manner in which natural language input to a video retrieval system is parsed using dependency trees, and the way in which the parse is refined to accommodate increasing levels of detail, until finally it can be matched with corresponding database content that can be presented as a viewable video sequence.

Index Terms: natural language dialogue, natural language understanding

1. Introduction

One of the objectives of the EPSRC-funded ISIS project (EP/E028640/1) is to enable supervisors in a network operations centre (NOC) to retrieve, by spoken enquiries, video information that has been captured and tagged by an intelligent sensor network. Processing of spoken enquiries falls to the natural language subcomponent of the ISIS project, ISIS-NL. Such a network, while operating primarily in the visual domain as a CCTV system, may also have the ability to capture environmental information in the acoustic and radio frequency domains. This information, describing the appearance of people and things, their interactions with each other and the environment, is likely in the longer term, to mirror the subtlety of a human observer's description of a scene or incident, and go beyond this in some modalities.

Accordingly we have adopted a highly flexible approach to the parsing of queries, and the storage of information and its retrieval. At the heart of this approach is an attempt to capture the semantic relationships between events and the actors that are involved in them, and between events or actors and the attributes or properties that describe them. The generation of dependency trees that represent such relationships characterizes the parsing process, and serves as a means of incorporating into the interpretative process a typical human appreciation of syntax and semantics: who or what typically performs a particular action; how is that action performed; to whom or to what is that action typically done. (In the discussion that follows we distinguish, in cases where ambiguity might arise, between 'logical objects', in the sense of object-oriented types, and 'grammatical objects', which suffer the action of some event.)

Similarly, the design of the normalized database in which tagged information is held – though it will not be the focus of this paper – also accommodates the many potential interrelationships and interactions between people, things, and their observed attributes (details which, incidentally, may rapidly and arbitrarily change during a recording period, not least because of the imperfections of automated recognition technologies.)

The process of mapping a parse tree to video clips consists of two major steps: firstly, identifying the parse pattern, and secondly locating and retrieving information related to an identified parse tree. The following sections describe these basic processes in greater detail, placing particular emphasis on the parsing process itself.

2. Identifying the parse pattern

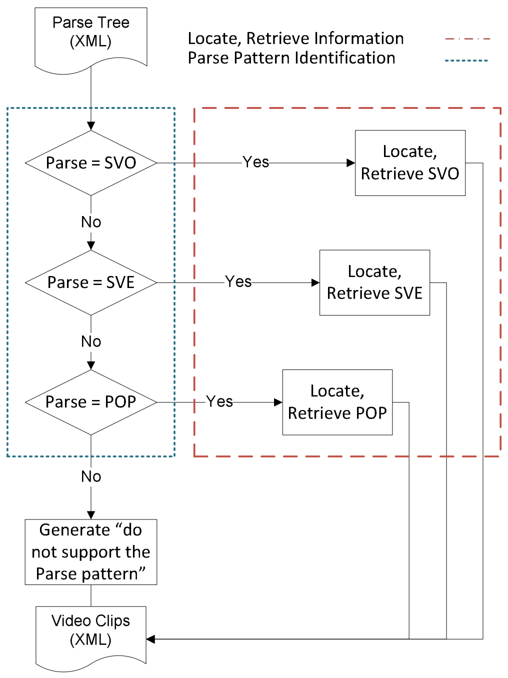

Figure 1. The process of mapping parse tree to video clips. The process comprises two steps: 'Identify the Pattern of Parse', and 'Locate and Retrieve Information'. If the process is faced with an unknown parse pattern, it may generate a basic response, or ask the dialogue manager to become involved.

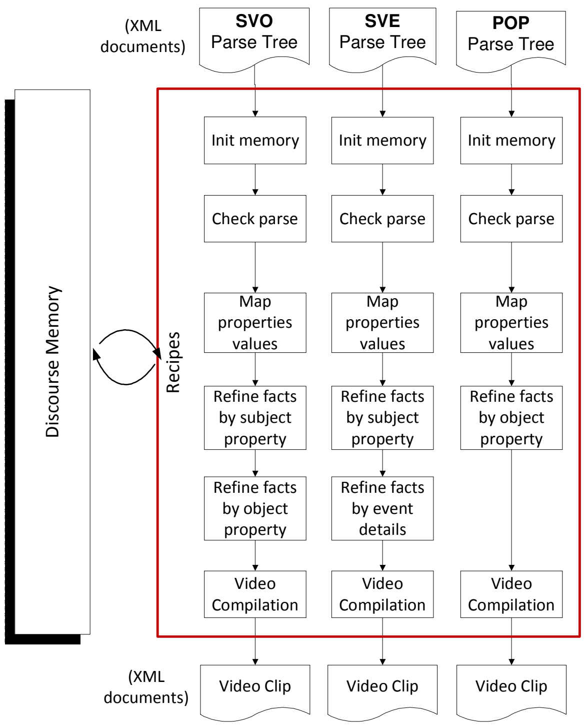

Figure 2. Architecture for locate and retrieve procedure

In ISIS-NL, the semantic parse of a natural language utterance is represented as a tree comprising a head and two other sub-trees: a left-hand tree and right-hand tree. The left and right sub-trees appear either in the form of a list (representing, for example, multiple subjects of an action or multiple properties of an entity) or in the form of another tree, again with a head and two other sub-trees. To map an utterance directly on to the underlying data scheme, a parse tree needs to be in one of the following forms:

- **SVO Pattern:** SVO refers to the utterances with the familiar *subject* + *verb* + *object* grammatical structure, e.g. "A young man in a red jacket got out of a yellow vehicle." Here, if we consider "got out of" as the verb (a prepositional verb interpreted as a transitive one), then the subject phrase is 'a young man in a red jacket', and the object is the 'yellow vehicle'.
- **SVE Pattern:** The SVE pattern refers to utterances in the form of a *subject* (agent) + *verb* + *event details* (adverbials), e.g. "A girl in a reddish hood walked somewhere". Here 'somewhere' is interpreted as a location adverbial that modifies the 'walking event' performed by the girl. It is often possible to interpret an SVO Pattern in terms of an SVE pattern and vice versa. Different ontological definitions of events results in different semantic parse rules that have to be considered, and different ways of representing an event in the data model. Study of [1] and [2] suggests such flexibility.
- **POP Pattern:** The POP pattern refers to utterances like "a young girl in a reddish hood": *property* + logical *object* + *property*. Here the head of parse tree is the entity 'girl', who is 'young' and she is 'in a reddish hood'. In the parse tree for this example both 'young' and 'in a reddish hood' are dependant on the 'girl' who is the head of the parse tree (In reality 'girl' would be normalized to the canonical form of the database – young, female person – and the object type 'person' would be further situated within an object hierarchy.)

3. Locating and retrieving information relevant to an identified parse tree

ISIS-NL uses the relationship between parsed entities – their semantic roles – for information retrieval. Thus, each of the parse patterns described previously introduces its own sequence of processes (sometimes variations on a common set of steps) that locate and retrieve information. Such a sequence of processes is known as a recipe. Figure 1 shows, from left to right, the recipes for the SVO, SVE, and POP Patterns. As can be seen from Figure 2, shared steps include memory initialization, locating information through a 'check parse' process, refining facts, and finally producing a video compilation. The recipe for each parse pattern can be represented by an XML entity, also called a *recipe*, each of whose steps has an equivalent predicate in Prolog, our main implementation language. A 'task specific discourse memory' evolves in parallel to the retrieval task: discourse memory contains facts that are the outcome of each step of a recipe. At the end of the retrieval task, discourse memory contains facts that are answers to an input query.

The main processes involved in each step of an information retrieval recipe are described in the following sections.

3.1.1. Init memory

The predicate *init_memory* initializes discourse memory, preparing it to hold information relevant to the final answer. Such information usually consists of pointers to facts about objects and events that are available through the data repository. The predicate *init_memory* is closely tied to the system's overall dialogue strategy, especially insofar as it concerns 'task-specific dialogue' and 'task-specific discourse memory' in a broader, potentially multi-domain dialogue system. For different real-world tasks, different sets of task-related values have to be initialized. However, often generic tactics (for deciding when to reset values, or for confirming new or changed values, etc.) will be employed by the dialogue manager (DM), as it initialises and subsequently manages the evolution and confirmation of facts supplied by the user, even if these are in a task-specific context.

3.1.2. Check parse

The *check_parse* predicate identifies information in the parse tree, and provides a list of events and objects as candidate answers. Candidate events and objects are chosen on the basis of the class of object and the type of event requested in the input query. The procedure may also consider the position of an event type or an object class in an 'event ontology' or an 'object ontology' respectively.

Figure 3 represents the flowchart for *check_parse* and its subgoals. Predicates *check_non_root*, *check_root*, and *check_object_sub_tree* respectively assert into memory all relevant event details and their semantic roles, all possible subjects or objects of events and their relevant properties, and all properties relevant to an object. For example, for an input query like "Did someone get out of a car?", *check_parse* finds that event 'get out of' is the most granular information that can appear as the root of the parse tree. It then finds and

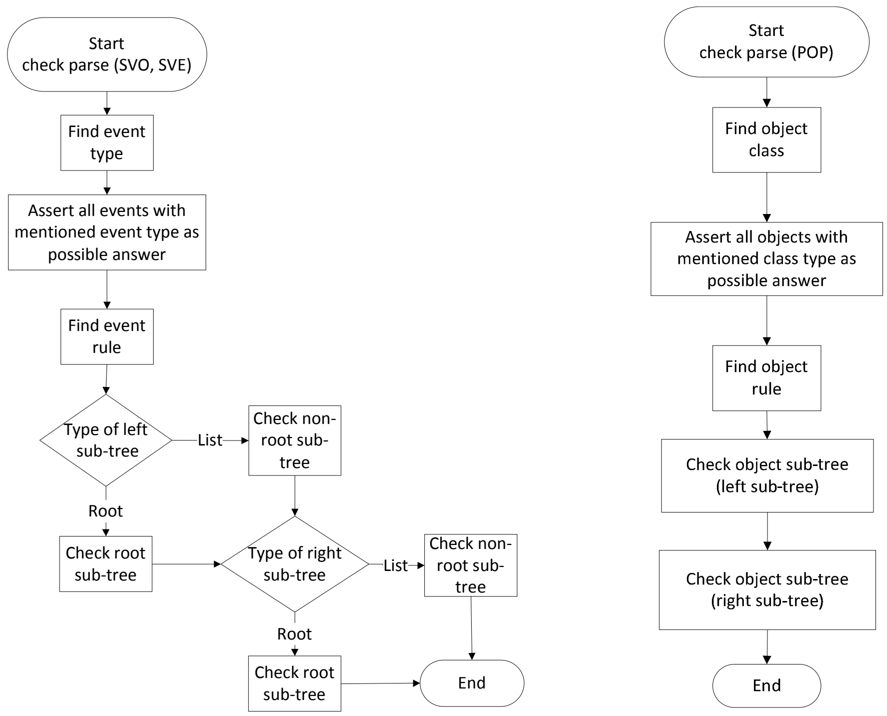

Figure 3. The *check_parse* procedure. The flowchart on the left shows *check_parse* for SVO and SVE parse patterns. The flowchart to the right illustrates the process for the POP parse pattern.

asserts in a data repository all the relevant events that are of type of 'get out of'. As for the next step, *check_parse* begins to look for the left and right sub-trees that it normally associates with an event such as 'get out of' and, using its parse rules (Figure 4), finds that a 'get out of' event needs a subject of class 'person' and a grammatical object of class 'container'. Using the information in the database, *check_parse* identifies logical objects that satisfy these criteria, and by asserting new Prolog facts in memory, identifies the objects and their semantic roles as possible answers. If the user's input contains properties, as descriptions of objects or as event details, *check_parse*, using ontology-based rules relating to events or objects, first ensures that the properties mentioned are compatible with the object class or event type, and then asserts the properties and their roles into memory, as candidate answers.

If the information in the input parse tree conflicts with a 'parse rule' (for example, a parse for 'get out of' may require that a *person* exits a *location*, rather than the other way round!), the system suspends further processing of the parse tree and generates a message to inform the broader dialogue system of the illogical combination of information. Depending on the particular implementation, a 'confirmation agent' may then intervene to solve the conflict. The system also asserts user-provided facts (obtained by the system's parse rules from the input parse tree) in its 'discourse memory', so that the broader dialogue system can deal with over-specified and under-specified queries and, in the latter case, ask the user to provide additional information as necessary.

3.1.3. Map property values

The predicate *map_property_values* maps the natural language description of a property to a list of equivalent, 'canonical' values that are used in the data repository. For example, 'black' as a description for the colour property may be mapped on to (0,0,0) as its RGB value. In the current implementation, mapping is fairly simple, and involves two steps: creating an atomic representation of the input property from the list containing the natural language description, and checking the atom against the vocabulary contained in the relevant property ontology. If the mapping procedure fails, then a fact to this effect is asserted into the discourse memory, to give the broader system a means of knowing about and responding to the problem.

3.1.4. Refine facts

The predicate *refine_facts* checks whether appropriate properties are associated with objects or events at particular times. The objects and events, which have previously been parsed and extracted from the user's utterance, are checked against properties so that appropriate object-property or event-property combinations may be identified as existing in the system's database. Corresponding to the entities that are capable of possessing properties, and that appear in the three varieties of parse trees, four variants of *refine_facts* have been implemented. In reality the number of *refine_facts* variants depends on the number of semantic roles supported by the system. In the current implementation, depending on the variant, *refine_facts* will look for database matches using

- the properties of the subject of an event;
- the properties of the object of an event;
- the properties of an object;
- the details of an event.

For example, in a query like "Did someone in a grey coat get out of a red car", 'grey coat' and 'red' are treated as

```
<event_rules type="get_out_of">
  <left expect="root"
    cat="ObjectPhrase" type="person"/>
  </left>
  <right expect="root">
    <cat=" ObjectPhrase" type="container"
  </right>
</event_rules>
```

```
<object_rule object_class="person">
  location
  gender
  height
  upper_garment
  headwear
  lower_garment
  hair
  time_tag
  age
</object_rule>
```

Figure 4. Parse Rules. On the left is an example of a parse rule for a 'get out of' event (an event_rule). It shows that this event can have a logical object of type 'person' as the grammatical subject of the event, and a logical object of type 'container' as the grammatical object. The subject appears in the left-hand sub-tree and the object in the right-hand one. The parse rule (object_rule) on the right of the figure tells us that a logical object of type person may have a list of dependant properties comprising age, location, gender, and so on. A container would have a list of properties too.

properties that describe the grammatical subject and object of a 'get out of" event. Let us assume that in the *check_parse* process we have already asserted persons *p1*, *p2*, *p3* and *p4* as possible subjects – because they are of the class *person* – of the event 'get out of". Similarly let us assume that vehicle *v1*, which is of class *container* in our system, is the only possible object of the event 'get out of'. The predicate *refine_facts* will assert all the time instances that *p1*, *p2*, *p3* and *p4* hold the property 'grey coat', and all the time instances that *v1* holds property 'red' – again, we take this rather cautious, and logical, approach, since an automated vision system may in fact assign differing and sometimes conflicting properties to the same object at different sampling times. In this example we may find that only *p3* has 'grey coat' as one of its properties and that 'red' is indeed a property for the object *v1*. As the result of the *refine_facts* procedure, previously asserted candidate answers that do not pass the refinement criteria are omitted from the list of candidate answers. After *refine_facts*, the discourse memory contains only facts that pass the relevant 'refinement' tests, and these facts are now augmented with temporal tags to assist retrieval and to inform the user *when* the candidate event or events occurred.

3.1.5. Video compilation

Having ensured, through the *refine_facts* procedure, that its candidate answers meet the criteria stipulated in the input query, the system now uses the predicate *video_compilation* and the temporal pointers that accompany the candidate answers to collect the key frames in the video repository that satisfy the user's request. Thresholds can be set on the number of frames collated, to ensure that the system presents a video sequence long enough to demonstrate the occurrence requested but not so long as to overburden the user. For events, a minimum of two key frames are used to show the time interval in which the event occurred. For objects, video is an assemblage of sets of time instants (with a minimum of one time instant per object). Thresholds can of course be adjusted to meet user requirements. The output of the *video_compilation* process is an XML file that represents video clips in the form of URLs for key frames. A simple animator front-end plays these back to the user in the form of a movie.

4. Conclusions

The end-to-end process described here concentrates on just one aspect of the dialogue process, namely the ability to match key content of a user's utterance with key content in the system's data repository. Of course, a fully developed dialogue system must do much more than this: it must ensure that the key contents of a user's utterance is properly understood; that changes and enhancements to the user's request are noted; that reasonable alternatives are presented to the user when a specific request cannot be fulfilled; and so on. We hope to address these matters in the coming months in the context of the ISIS project, drawing on our previous experience of developing the Queen's Communicator [4] dialogue system. For the moment, though, we believe we have devised a useful approach to handling the uncertainties that arise from 'hard-to-recognise' and 'hard-to-interpret' visual information. Our approach accommodates and deals with possibly erroneous classifications by an intelligent vision system, and makes use of human-level knowledge, embodied in grammatically- and semantically-based parse rules, to help impose reasonable interpretations both on a user request and the on the information that might be used to satisfy it.

5. References

[1] Smith, J. O. and Abel, J. S., "Bark and ERB Bilinear Transforms", IEEE Trans. Speech and Audio Proc., 7(6):697-708, 1999.

[2] Soquet, A., Saerens, M. and Jospa, P., "Acoustic-articulatory inversion", in T. Kohonen [Ed], Artificial Neural Networks, 371-376, Elsevier, 1991.

[3] Stone, H.S., "On the uniqueness of the convolution theorem for the Fourier transform", NEC Labs. Amer. Princeton, NJ. Online: http://citeseer.ist.psu.edu/176038.html, accessed on 19 Mar 2008.

[4] O'Neill, I., Hanna, P., Liu, X. and McTear, M., "The Queen's Communicator: An Object-Oriented Dialogue Manager", Proc. EUROSPEECH-2003, Geneva, pp. 593-596, 2003.