$RESTAURANT re^n-visited: A KM Implementation of a Compositional Approach Working Note 17

Peter Clark
Knowledge Systems
Boeing, PO Box 3707, Seattle, WA 98124
peter.e.clark@boeing.com

Bruce Porter
Department of Computer Science
University of Texas at Austin, TX 78712
porter@cs.utexas.edu

March 2000

Abstract

This Working Note has two distinct goals. First, it presents an example KM representation showing how a detailed script can be constructed by combining more general scripts. In particular, it illustrates how interactions *between* those general scripts, as well as between each general script and the target detailed script, can be specified. The significance of this note is that it presents these ideas at the implementation level, missing from earlier expositions (by us and others) of composition.

Second, as an independent goal, it shows the representation expressed both using KM classes, and KM prototypes, to provide a side-by-side comparison of these different representational styles using the KM language. In fact, these two KBs are extremely similar; our point is thus not to suggest a major change in how knowledge is expressed, but to suggest that with relatively trivial tweaks in that expression (changing classes to prototypes), there may be significant advantages for inter-language portability, graphical display, and dissemination of our component library, in a way analogous to WordNet's success.

1 Introduction

This working note has two rather distinct goals. First, drawing from a familiar example in AI, we present sample KM code showing how a restaurant visit can be described as a composition of two more general scripts (purchasing and dining). The idea of building scripts compositionally is, of course, not new: Schank, dissatisfied with the rigidity of his original script idea, later strongly advocated that scripts should be build compositionally out of fragments ("MOPs") [Schank, 1982]; similarly, Dyer sketched out some general ideas on how this composition might look [Dyer, 1981]. We have also been pushing on this idea in our work on composition [Clark and Porter, 1997, Clark and Porter, 1995], as have others [Chapman, 1986, Noy and Hafner, 1998, Levy, 1993]. This Working Note presents a concrete KM implementation of these ideas, and in particular shows how extra interconnections between components – in this case, constraints on the ordering of events in the script (e.g., Do you pay before or after eating?) – can be specified. While KM's unification algorithm can combine much of the component structures automatically, there may still be ambiguity or missing information about how the information should merge

to form a final representation (for example again, do you pay before or after eating in a restaurant?). In fact, interconnecting components with each other in different ways can give rise to different compositions. We have not previously illustrated how to specify these extra interconnections, but do so here.

A second, and independent, goal of this paper is to show the representation expressed in two different styles, one using traditional KM classes, and one using KM prototypes[1]. We assume familiarity with KM and these representational styles, and do not repeat a description here. (For details, see the KM User's Manual [Clark and Porter, 1999]). In fact, the two KBs (using classes and using prototypes) are syntactically very similar – so much so, that the reader may ask what the point is in suggesting prototypes when they look essentially the same as the class-based KB. In fact, the main point of prototypes is not in their external appearance, but in the way they are stored and manipulated internally. For the class-based representation, KM stores and reasons with it exactly as the knowledge engineer has expressed it, while for the prototype-based representation, KM "compiles" each prototype into a simple graph of instances (which we refer to as a "graphlet"), and then reasons with those through a process of graph cloning and unification. Section 5 shows a (machine-generated) dump of the graphlets of these prototypes, to make this internal form explicit. This graphlet database has some similarity in style to WordNet [Miller et al., 1993], whose simplicity and portability have made it a widely used resource. It raises the interesting idea of eventually publishing an analagous resource ("KnowledgeNet", say), namely a large database of graphlets that others could use compositionally in other projects. Graphlets also have the appeal of providing direct data for a graph manipulation interface to operate on, such as the WebMod interface being considered for DARPA's RKF project.

The point of the class/prototype comparison here is thus not to suggest a major change in how knowledge is expressed, but to suggest that with relatively trivial tweaks in that expression (changing classes to prototypes), there may be significant advantages for inter-language portability, graphical display, and dissemination of our component library.

This Working Note is meant as a simple illustration of script composition. For a more complex example (in the domain of molecular biology) see the subsequent Working Note to this [Clark and Porter, 2000]. The biology example is essentially identical in style to the one in this Working Note, just with more details and compositions involved.

We now present the representation and its two implementations, the database of graphlets produced from the prototypes, and a fragment of the WordNet database to suggest similarity of graphlets with WordNet.

2 The Restaurant Script

2.1 Representation

2.1.1 The Generic Restaurant Script

This toy example models a restaurant script as follows:

`Restaurant-Visiting` = a composition of a `Purchasing` and a `Dining`.

`Purchasing` = a `Getting` and a `Paying` (order unspecified).

[1] "Prototypes" is somewhat of a misnomer, and the name will probably changed in future.

`Dining` = a `Sitting`, a `Getting`, and `Eating` (with the eating after the sitting and after the getting).

We consider `Getting` and `Paying` to be subevents of `Purchasing`, but we consider `Purchasing` and `Dining` to be 'component-events' of `Restaurant-Visiting`, because they are both ongoing (and interleave) throughout the whole restaurant visit. We use two different predicate (slot) names (`subevents` and `component-events`) to distinguish these two subtly different relationships.

Also of interest is that the two `Getting`s above are coreferential, both referring to the event where the food is tranferred from the (server in the) restaurant to the diner. (For simplicity we do not model the server explicitly, and instead consider the restaurant to be the 'server'). Thus we need to specify this coreference in the specification of `Restaurant-Visiting`, to avoid the diner getting his/her food twice.

2.1.2 Specializations of Restaurant Visit

We then describe two subclasses of `Restaurant-Visiting`, namely visiting a McDonalds restaurant and visiting the (fancier) Trudys restaurant. These specialize the restaurant visit script by providing additional ordering information on the subevents: In a McDonalds visit, the diner pays before being given the meal, which is before he/she sits down to eat. In a Trudy's visit, the diner sits before he/she eats, and eats before he/she pays. To specify this information, these specializations of restaurant visit need to refer to these subevents, introduced in the generic restaurant visit. This is done by what we sometimes call "shadowing", namely repeating the concepts (here component events) we want to refer to, and then adding qualifications. This is shown (in a simplified form) below for the class-based knowledge-base (Section 3 shows the full KB):

```
;;; (Part of) the generic restaurant concept
(every Restaurant-Visiting has
  (component-events ((a Dining) (a Purchasing))))

;;; Now repeat ("shadow") the fact that a McDonalds restaurant visit also has a
;;; Dining, so that we can then qualify it (namely, its Getting is before the Eating):
(every McDonalds-Restaurant-Visiting has
  (component-events ((a Dining with                  ; repeat (``shadow'')...
                       (subevents ((a Getting with    ; ... then qualify
                                     (before ((the Eating subevents of
                                                (the Dining component-events
                                                   of Self)))))))))))
```

Similarly, in a prototype we must again re-introduce the objects we want to qualify before we can refer to them:

```
;;; (Part of) the generic restaurant concept
(a-prototype McDonalds-Restaurant-Visiting)

((the Restaurant-Visiting) has
  (component-events ((a Dining) (a Purchasing))))

(end-prototype)

;;; Now repeat ("shadow") the fact that a McDonalds restaurant visit also has a
;;; Dining, so that we can then qualify it (namely, its Getting is before the Eating):
```

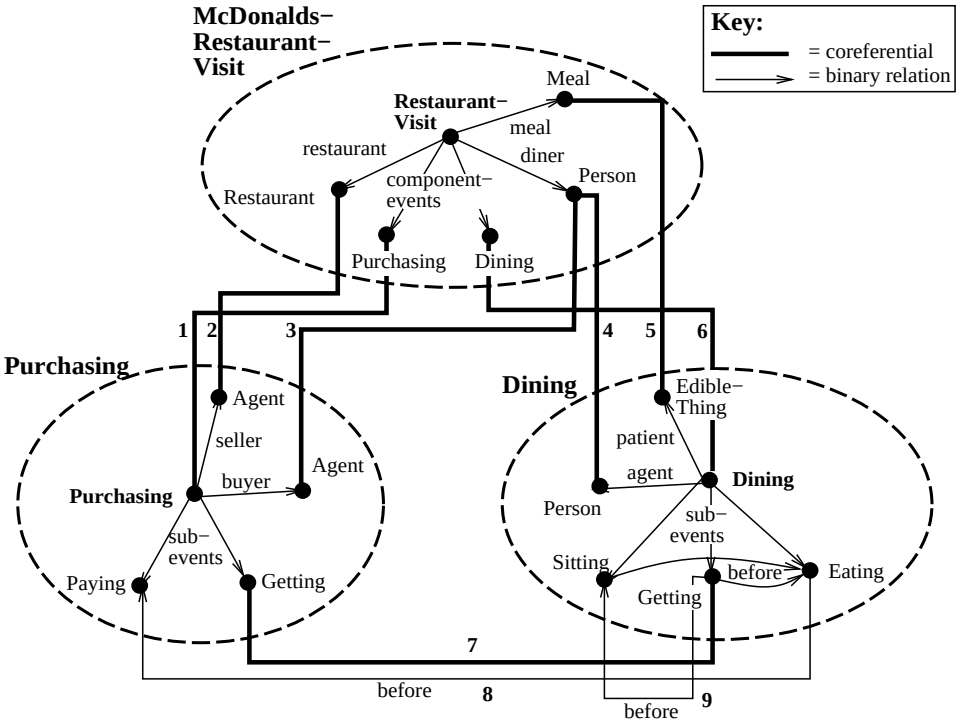

Figure 1: The specification of a McDonalds Restaurant Visit requires not only knowledge of how a restaurant visit is made up from its components (purchasing and dining), but also how those components interconnect. Connections 1-7 are generic to all restaurant visits, and so are specified as part of the `Restaurant-Visiting` script. Connections 8 and 9 are specific to (here) McDonalds restaurant visits, and so specified in the `McDonalds-Restaurant-Visiting` script. See Sections 3 and 4 for the KM implementation which this sketch illustrates.

```
(a-prototype McDonalds-Restaurant-Visiting)

((the McDonalds-Restaurant-Visiting) has
   (component-events ((a Dining with                  ; repeat (‘‘shadow’’)...
                         (subevents ((a Getting) (a Eating)))))))

((the Getting) has (before ((the Eating))))           ; ... then qualify

(end-prototype)
```

This repetition of structure in the more specific classes/prototypes ("shadowing") is not ideal, but seems to be the only way to get a handle on those objects that we want to assert something about.

The construction of a McDonalds restaurant visit from components is sketched (with some details missing) in Figure 1, showing both how the generic restaurant visit is itself constructed from components (purchasing and dining, with connections 1-7), and how the McDonalds restaurant visit further refines this (by adding relations 8 and 9). Sections 3 and 4 show the KM implementation which this sketch summarizes.

Finally, note that there are many other aspects of visiting a restaurant which we have not modeled here (e.g. the goals or intensions of the various parties involved). Our goal here is not detail, but to illustrate the general mechanism of composition at the implementation level.

2.2 The Performance Task

This representation is designed to answer just a single question, namely "What are the subevents of <*some main event*>?". KM constructs a list of events (KM instances), with their roles appropriately filled in, and with appropriate constraints on their ordering. In principle, if we added representations for these primitive events themselves (e.g., a STRIPS-like description of which facts are made true/false by the events), then KM could perform a simulation of the scenario by "executing" each action in turn. This would result in a sequence of KM situations in the knowledge-base, each describing the state of the world at a different point in the script. From here, additional questions could be answered, posed to situations in this sequence. We have not taken this example this far.

Both versions of the representation below are able to answer our target competency question (for different restaurants), namely:

```
;;; "What are the subevents in a visit to a <particular type of> restaurant?"
KM> (the all-subevents of (a McDonalds-Restaurant-Visiting))
```

This query will just return a set of subevents. To also display the ordering information associated with these events, we can ask KM to find out the before/cotemporal-with/after information associated with each subevent, and print it out. To do this, we put a (rather ugly) collect-and-print KM expression on the `description-of-all-subevents` slot of the `Event` frame (see Sections 3 and 4), which gathers and prints out this information.

Given this, the following shows a trace of KM answering the performance question:

```
;;; "What are the subevents in a (generic) restaurant visit?"
KM> (the description-of-all-subevents of (a Restaurant-Visiting))
The paying.
The getting. The getting is before the eating.
The sitting. The sitting is before the eating.
The eating. The eating is after the sitting the getting.

;;; "What are the subevents in a McDonalds restaurant visit?"
KM> (the description-of-all-subevents of (a McDonalds-Restaurant-Visiting))
The paying. The paying is before the getting.
The getting. The getting is before the sitting the eating.
             The getting is after the paying.
The sitting. The sitting is before the eating. The sitting is after the getting.
The eating. The eating is after the getting the sitting.

;;; "What are the subevents in a Trudys restaurant visit?"
KM> (the description-of-all-subevents of (a Trudys-Restaurant-Visiting))
The sitting. The sitting is before the eating the getting.
The getting. The getting is before the eating. The getting is after the sitting.
The eating. The eating is before the paying.
            The eating is after the getting the sitting.
The paying. The paying is after the eating.
```

The above output shows that KM has inferred that the two different restaurant visits have two different orderings of their subevents:

McDonalds: Paying →Getting →Sitting →Eating

Trudys: Sitting →Getting →Eating →Paying

In addition, although not revealed by this print-out, each subevent contains information about who its players are (agent, patient, etc.). The knowledge-base could thus answer questions about these facts also.

Note that the order in which KM prints out the actions (above) is irrelevant; rather, the ordering information is contained in the before/cotemporal-with/during constraints. For convenience the above output has been manually edited to list the actions in the order which the constraints specify. A simple procedure could perform such linearization automatically.

We now show the knowledge bases themselves.

3 KM Representation: Using Classes

```
;;; File: restaurant-classes.km
;;; Date: March 2000

;;; This file requires KM 1.4.0-beta33 or later

(reset-kb)

;;; cotemporal-with isn't used in this KB, but we put it there anyway for completeness
(before has (instance-of (Slot)) (inverse (after)))
(cotemporal-with has (instance-of (Slot)) (inverse (cotemporal-with)))
(subevents has (instance-of (Slot)) (inverse (superevents)))

;;; [1] This ugly formatting statements simply prints out the before, cotemporal-with, and after
;;; properties of for each subevent of the main event.
(every Event has
  (all-subevents ((the subevents of Self)
                  (the all-subevents of (the subevents of Self))))
  (subevents ((the subevents of (the component-events of Self))))
  (description-of-all-subevents (
      (make-sentence
             (forall (the all-subevents of Self)                        ; [1]
                (:seq It "."
                      (if   (has-value (the before of It))
                       then (:seq It "is before" (the before of It) "."))
                      (if   (has-value (the cotemporal-with of It))
                       then (:seq It "is cotemporal with" (the cotemporal-with of It) "."))
                      (if   (has-value (the after of It))
                       then (:seq It "is after" (the after of It) "."))
                      (format nil "~%")))))))

(Paying has (superclasses (Event)))
(Sitting has (superclasses (Event)))
(Getting has (superclasses (Event)))
(Eating has (superclasses (Event)))

;;; ----------------------------------------
;;;             PURCHASING
;;; ----------------------------------------

(Purchasing has (superclasses (Event)))

(every Purchasing has
  (buyer ((a Agent)))
  (seller ((a Agent)))
  (item ((a Thing)))
  (money ((a Amount-Of-Money)))
  (subevents (                          ; NOTE no ordering assumed here
    (a Getting with
       (agent ((the seller of Self)))
       (patient ((the item of Self)))
       (recipient ((the buyer of Self))))
    (a Paying with
       (agent ((the buyer of Self)))
       (patient ((the money of Self)))
       (recipient ((the seller of Self)))))))

;;; ----------------------------------------
```

```
;;;             DINING
;;; ----------------------------------------

(Dining has (superclasses (Event)))

(every Dining has
   (agent ((a Person)))
   (patient ((a Edible-Thing)))
   (location ((a Place)))
   (subevents (                                ; again, no ordering implied
     (a Sitting with
       (agent ((the agent of Self)))
       (location ((the location of Self)))
       (before ((the Eating subevents of Self))))
     (a Getting with
       (agent ((the agent of Self)))
       (patient ((the patient of Self)))
       (before ((the Eating subevents of Self))))
     (a Eating with
       (agent ((the agent of Self)))
       (patient ((the patient of Self)))
       (after ((the Sitting subevents of Self)
               (the Getting subevents of Self)))))))

;;; ----------------------------------------
;;;     THE COMPOSITION: RESTAURANT VISITING = PURCHASING + DINING
;;; ----------------------------------------

(Restaurant-Visiting has (superclasses (Event)))

;;; [1] This is a slightly awkward way of saying the two Gettings (in the Purchasing and Dining)
;;; are coreferential.
(every Restaurant-Visiting has
  (diner ((a Person)))
  (meal ((a Meal)))
  (restaurant ((a Restaurant)))
  (table ((a Table with
            (location ((the restaurant of Self))))))
  (money ((a Amount-Of-Money)))
  (component-events (
    (a Purchasing with
      (buyer ((the diner of Self)))
      (seller ((the restaurant of Self)))
      (item ((the meal of Self)))
      (money ((the money of Self)))
      (subevents ((the Getting subevents of (the Dining component-events of Self))))) ; [1]
    (a Dining with
      (agent ((the diner of Self)))
      (patient ((the meal of Self)))
      (location ((the table of Self)))
      (subevents ((the Getting subevents of (the Purchasing component-events of Self))))))))    ; [1]

;;; ----------------------------------------
;;;     MCDONALDS-RESTAURANT-VISITING: Extra constraints on event ordering specified
;;; ----------------------------------------

(McDonalds-Restaurant-Visiting has (superclasses (Restaurant-Visiting)))

;;; We now define the McDonalds-Specific Connections between components:
```

```
;;; [1] the Paying is before the Getting.
;;; [2] the Getting is before the Sitting and the Eating.
(every McDonalds-Restaurant-Visiting has
   (component-events (
      (a Purchasing with
        (subevents ((a Paying with
                        (before ((the Getting subevents of              ; [1]
                                    (the Dining component-events of Self))))))))
      (a Dining with
        (subevents ((a Getting with
                        (before ((the Sitting subevents of              ; [2]
                                    (the Dining component-events of Self))
                                 (the Eating subevents of
                                    (the Dining component-events of Self)))))))))))

;;; ----------------------------------------
;;;     TRUDYS-RESTAURANT-VISITING: Extra constraints on event ordering specified
;;; ----------------------------------------

(Trudys-Restaurant-Visiting has (superclasses (Restaurant-Visiting)))

;;; [1] the Paying is after the Eating.
;;; [2] the Getting is after the Sitting.
(every Trudys-Restaurant-Visiting has
   (component-events (
      (a Purchasing with
        (subevents ((a Paying with
                        (after ((the Eating subevents of                ; [1]
                                    (the Dining component-events of Self)))))
                    (a Getting with
                        (after ((the Sitting subevents of               ; [2]
                                    (the Dining component-events of Self)))))))))))

;;; --- end ---
```

4 KM Represesentation: Using Prototypes

```
;;; File: restaurant-prototypes.km
;;; Date: March 2000

;;; This file requires KM 1.4.0-beta33 or later

(reset-kb)

;;; cotemporal-with isn't used in this KB, but we put it there anyway for completeness
(before has (instance-of (Slot)) (inverse (after)))
(cotemporal-with has (instance-of (Slot)) (inverse (cotemporal-with)))
(subevents has (instance-of (Slot)) (inverse (superevents)))

;;; [1] This ugly formatting simply prints out the before, cotemporal-with, and after properties of
;;; for each subevent of the main event.
(every Event has
  (all-subevents ((the subevents of Self)
                  (the all-subevents of (the subevents of Self))))
  (subevents ((the subevents of (the component-events of Self))))
  (description-of-all-subevents (
      (make-sentence
             (forall (the all-subevents of Self)                        ; [1]
                (:seq It "."
                      (if   (has-value (the before of It))
                       then (:seq It "is before" (the before of It) "."))
                      (if   (has-value (the cotemporal-with of It))
                       then (:seq It "is cotemporal with" (the cotemporal-with of It) "."))
                      (if   (has-value (the after of It))
                       then (:seq It "is after" (the after of It) "."))
                      (format nil "~%")))))))

(Paying has (superclasses (Event)))
(Sitting has (superclasses (Event)))
(Getting has (superclasses (Event)))
(Eating has (superclasses (Event)))

;;; ----------------------------------------
;;;             PURCHASING
;;; ----------------------------------------

(Purchasing has (superclasses (Event)))

(a-prototype Purchasing)

((the Purchasing) has
  (buyer ((a Agent)))
  (seller ((a Agent)))
  (item ((a Thing)))
  (money ((a Amount-Of-Money)))
  (subevents (                          ; NOTE no ordering assumed here
    (a Getting with
       (agent ((the seller of Self)))
       (patient ((the item of Self)))
       (recipient ((the buyer of Self))))
    (a Paying with
       (agent ((the buyer of Self)))
       (patient ((the money of Self)))
       (recipient ((the seller of Self)))))))
```

```
(end-prototype)

;;; ----------------------------------------
;;;             DINING
;;; ----------------------------------------

(Dining has (superclasses (Event)))

(a-prototype Dining)

((the Dining) has
   (agent ((a Person)))
   (patient ((a Edible-Thing)))
   (location ((a Place)))
   (subevents (                                ; again, no ordering implied
     (a Sitting with
       (agent ((the agent of Self)))
       (location ((the location of Self))))
     (a Getting with
       (agent ((the agent of Self)))
       (patient ((the patient of Self))))
     (a Eating with
       (agent ((the agent of Self)))
       (patient ((the patient of Self)))))))

((the Sitting) has (before ((the Eating))))
((the Getting) has (before ((the Eating))))
;;; (but we don't know whether the sitting is before or after the getting)

(end-prototype)

;;; ----------------------------------------
;;;     THE COMPOSITION: RESTAURANT VISITING = PURCHASING + DINING
;;; ----------------------------------------

(Restaurant-Visiting has (superclasses (Event)))

(a-prototype Restaurant-Visiting)

((the Restaurant-Visiting) has
  (diner ((a Person)))
  (meal ((a Meal)))
  (restaurant ((a Restaurant)))
  (table ((a Table with
            (location ((the restaurant of Self))))))
  (money ((a Amount-Of-Money))))

;;; Specify the components...

((the Restaurant-Visiting) has
  (component-events (
    (a Purchasing with
      (buyer ((the Person)))
      (seller ((the Restaurant)))
      (item ((the Meal)))
      (money ((the Amount-Of-Money))))
    (a Dining with
      (agent ((the Person)))
```

```
      (patient ((the Meal)))
      (location ((the Table)))))))

;;; Introduce these subevents (as I want to then refer to them)...
((the Purchasing) has (subevents ((a Getting))))
((the Dining) has (subevents ((a Getting))))

;;; ...and then state they are coreferential (== does unification)
((the Getting subevents of (the Purchasing)) ==  (the Getting subevents of (the Dining)))

(end-prototype)

;;; ----------------------------------------
;;;     MCDONALDS-RESTAURANT-VISITING: Extra constraints on event ordering specified
;;; ----------------------------------------

(McDonalds-Restaurant-Visiting has (superclasses (Restaurant-Visiting)))

(a-prototype McDonalds-Restaurant-Visiting)

;;; We now define the McDonalds-Specific Connections between components:
;;; (a) Explicitly create the to-be-referred-to objects...

((the McDonalds-Restaurant-Visiting) has
   (component-events ((a Purchasing with
                        (subevents ((a Getting) (a Paying))))
                      (a Dining with
                        (subevents ((a Sitting) (a Eating)))))))

;;; ...then (b) state the connections of interest...
((the Paying) has (before ((the Getting))))
((the Getting) has (before ((the Sitting))))
((the Getting) has (before ((the Eating))))

(end-prototype)

;;; ----------------------------------------
;;;     TRUDYS-RESTAURANT-VISITING: Extra constraints on event ordering specified
;;; ----------------------------------------

(Trudys-Restaurant-Visiting has (superclasses (Restaurant-Visiting)))

(a-prototype Trudys-Restaurant-Visiting)

;;; We now define the Trudys-Specific Connections between components:
;;; (a) Explicitly create the to-be-referred-to objects...

((the Trudys-Restaurant-Visiting) has
   (component-events ((a Purchasing with
                        (subevents ((a Getting) (a Paying))))
                      (a Dining with
                        (subevents ((a Sitting) (a Eating)))))))

;;; ...then (b) state the connections of interest...
((the Paying) has (after ((the Eating))))
((the Getting) has (after ((the Sitting))))

(end-prototype)
```

5 Graphlets for the KM Prototypes

The following shows a (automatically generated) dump of the graphlets for this example, i.e., KM's internal representation of the prototypes specified in Section 4. These graphlets are constructed by KM at load-time, when it loads the prototype descriptions in Section 4. (This print-out was generated by a small Lisp procedure, separate from the KM implementation). It is shown in Prolog-like syntax to emphasize similarity in style with WordNet, a fragment of which is shown in Section 6 for comparison. Graphlets have well-defined semantics: If graphlet G of concept C_0 contains instances $I_0, I_1, ..., I_n$ in corresponding classes $C_0, C_1, ..., C_n$, and relations $R = \{r(I_i, I_j)\}$, then for all instances I'_0 of C_0, there exists instances $I'_1, ..., I'_n$ of classes $C_1, ..., C_n$ such that relations $R' = \{r(I'_i, I'_j)\}$ hold, where R' is a copy of R with the instances $\{I_n\}$ replaced with their corresponding instances $\{I'_n\}$.

```
graphlet(0,'Purchasing').

participants(0,0).
participants(0,1).
participants(0,2).
participants(0,3).
participants(0,4).
participants(0,5).
participants(0,6).

isa(0,'Purchasing').
isa(1,'Paying').
isa(2,'Getting').
isa(3,'Amount-Of-Money').
isa(4,'Thing').
isa(5,'Agent').
isa(6,'Agent').

buyer(0,6).
seller(0,5).
item(0,4).
money(0,3).
subevents(0,2).
subevents(0,1).
agent(1,6).
patient(1,3).
recipient(1,5).
superevents(1,0).
agent(2,5).
patient(2,4).
recipient(2,6).
superevents(2,0).
'money-of'(3,0).
'patient-of'(3,1).
'item-of'(4,0).
'patient-of'(4,2).
'seller-of'(5,0).
'agent-of'(5,2).
'recipient-of'(5,1).
'buyer-of'(6,0).
'recipient-of'(6,2).
'agent-of'(6,1).
```

```
% ----------

graphlet(7,'Dining').

participants(7,7).
participants(7,8).
participants(7,9).
participants(7,10).
participants(7,11).
participants(7,12).
participants(7,13).

isa(7,'Dining').
isa(8,'Eating').
isa(9,'Getting').
isa(10,'Sitting').
isa(11,'Place').
isa(12,'Edible-Thing').
isa(13,'Person').

agent(7,13).
patient(7,12).
location(7,11).
subevents(7,10).
subevents(7,9).
subevents(7,8).
agent(8,13).
patient(8,12).
superevents(8,7).
after(8,9).
after(8,10).
agent(9,13).
patient(9,12).
superevents(9,7).
before(9,8).
agent(10,13).
location(10,11).
superevents(10,7).
before(10,8).
'location-of'(11,10).
'location-of'(11,7).
'patient-of'(12,8).
'patient-of'(12,9).
'patient-of'(12,7).
'agent-of'(13,8).
'agent-of'(13,9).
'agent-of'(13,10).
'agent-of'(13,7).

% ----------

graphlet(14,'Restaurant-Visiting').

participants(14,14).
participants(14,15).
participants(14,16).
participants(14,17).
participants(14,18).
```

```
participants(14,19).
participants(14,20).
participants(14,21).
participants(14,22).

isa(14,'Restaurant-Visiting').
isa(15,'Getting').
isa(16,'Dining').
isa(17,'Purchasing').
isa(18,'Amount-Of-Money').
isa(19,'Table').
isa(20,'Restaurant').
isa(21,'Meal').
isa(22,'Person').

diner(14,22).
meal(14,21).
restaurant(14,20).
table(14,19).
money(14,18).
'component-events'(14,17).
'component-events'(14,16).
superevents(15,17).
superevents(15,16).
agent(16,22).
patient(16,21).
location(16,19).
'component-events-of'(16,14).
subevents(16,15).
buyer(17,22).
seller(17,20).
item(17,21).
money(17,18).
'component-events-of'(17,14).
subevents(17,15).
'money-of'(18,17).
'money-of'(18,14).
location(19,20).
'table-of'(19,14).
'location-of'(19,16).
'restaurant-of'(20,14).
'location-of'(20,19).
'seller-of'(20,17).
'meal-of'(21,14).
'item-of'(21,17).
'patient-of'(21,16).
'diner-of'(22,14).
'buyer-of'(22,17).
'agent-of'(22,16).

% ----------

graphlet(23,'McDonalds-Restaurant-Visiting').

participants(23,23).
participants(23,24).
participants(23,25).
participants(23,26).
participants(23,27).
```

```
participants(23,28).
participants(23,29).

isa(23,'McDonalds-Restaurant-Visiting').
isa(24,'Eating').
isa(25,'Sitting').
isa(26,'Paying').
isa(27,'Getting').
isa(28,'Dining').
isa(29,'Purchasing').

'component-events'(23,29).
'component-events'(23,28).
superevents(24,28).
after(24,27).
superevents(25,28).
after(25,27).
superevents(26,29).
before(26,27).
superevents(27,29).
after(27,26).
before(27,25).
before(27,24).
subevents(28,25).
subevents(28,24).
'component-events-of'(28,23).
subevents(29,27).
subevents(29,26).
'component-events-of'(29,23).

% ----------

graphlet(30,'Trudys-Restaurant-Visiting').

participants(30,30).
participants(30,31).
participants(30,32).
participants(30,33).
participants(30,34).
participants(30,35).
participants(30,36).

isa(30,'Trudys-Restaurant-Visiting').
isa(31,'Eating').
isa(32,'Sitting').
isa(33,'Paying').
isa(34,'Getting').
isa(35,'Dining').
isa(36,'Purchasing').

'component-events'(30,36).
'component-events'(30,35).
superevents(31,35).
before(31,33).
superevents(32,35).
before(32,34).
superevents(33,36).
after(33,31).
superevents(34,36).
```

```
after(34,32).
subevents(35,32).
subevents(35,31).
'component-events-of'(35,30).
subevents(36,34).
subevents(36,33).
'component-events-of'(36,30).

% ----------
```

6 A Sample of WordNet

Finally, we show a tiny fragment of WordNet, to emphasize similarities with the graphlet database of Section 5. Our point here is that a database of graphlets could be considered as the next evolutionary step in the creation of a WordNet-like knowledge resource.

```
% ----------
% Concepts ("synsets") and their text description (gloss). Each concept is
% identified by a number in WordNet.

% g(ConceptId, TextDescription)
g(103233732,'(a building where people go to eat)').
g(102292428,'(a small informal restaurant; serves wine)').
g(102331449,'(a small restaurant serving beer and wine as well as food; usually cheap)').
g(102366126,'(a small restaurant where drinks and snacks are sold)').
g(102347413,'(a structure that has a roof and walls and stands more or less permanently in
             one place; "there was a three-story building on the corner")').
g(106008951,'(a chain of restaurants)').
g(102190772,'(an addition that extends a main building)').
g(102191831,'(a large entrance or reception room or area)').
g(102501991,'((architecture) solid exterior angle of a building; especially one formed by
             a cornerstone)').

% ----------
% Words (synonyms) used to refer to each concept

% s(ConceptId,SynonymNumber,Word,WordType,WordSenseNumber,Tag /*(not of interest)*/)
s(103233732,1,'restaurant',n,1,1).
s(103233732,2,'eating_house',n,1,0).
s(103233732,3,'eating_place',n,1,0).

s(102292428,1,'bistro',n,1,0).

s(102331449,1,'brasserie',n,1,0).

s(102366126,1,'cafe',n,1,1).
s(102366126,2,'coffeehouse',n,1,1).
s(102366126,3,'coffee_shop',n,1,1).
s(102366126,4,'coffee_bar',n,1,0).

s(102347413,1,'building',n,1,1).
s(102347413,2,'edifice',n,1,1).

s(106008951,1,'restaurant_chain',n,1,0).

s(102190772,1,'annex',n,1,0).
s(102190772,2,'annexe',n,1,0).
s(102190772,3,'extension',n,10,0).
s(102190772,4,'wing',n,8,0).

s(102191831,1,'anteroom',n,1,0).
s(102191831,2,'antechamber',n,1,0).
s(102191831,3,'entrance_hall',n,1,1).

s(102501991,1,'corner',n,11,0).
s(102501991,2,'quoin',n,3,0).

% ----------
```

```
% Generalization relationships (Hypernyms)

% hyp(SubConceptId,SuperConceptId)
hyp(102292428,103233732).
hyp(102331449,103233732).
hyp(102366126,103233732).
hyp(103233732,102347413).

% ----------
% Member-of relationships (member meronyms)
% mm(Member,Group)

mm(103233732,106008951).

% ----------
% Part-of relationships (part meronyms)
% mp(Part,Whole)

mp(102190772,102347413).
mp(102191831,102347413).
mp(102501991,102347413).
```

References

[Chapman, 1986] Chapman, D. (1986). Cognitive cliches. AI Working Paper 286, MIT, MA.

[Clark and Porter, 1995] Clark, P. and Porter, B. (1995). Working note 6: Constructing scripts from components. (http://www.cs.utexas.edu/users/pclark/working_notes).

[Clark and Porter, 1997] Clark, P. and Porter, B. (1997). Building concept representations from reusable components. In *AAAI-97*, pages 369–376, CA. AAAI. (http://www.cs.utexas.edu/users/pclark/papers).

[Clark and Porter, 1999] Clark, P. and Porter, B. (1999). KM – the knowledge machine: Users manual. Technical report, AI Lab, Univ Texas at Austin. (http://www.cs.utexas.edu/users/mfkb/km.html).

[Clark and Porter, 2000] Clark, P. and Porter, B. (2000). Working note 18: Constructing scripts compositionally: A microbiology example. (http://www.cs.utexas.edu/users/pclark/working_notes).

[Dyer, 1981] Dyer, M. G. (1981). $RESTAURANT revisited, or 'lunch with boris'. In *IJCAI-81*, pages 234–236.

[Levy, 1993] Levy, A. Y. (1993). Irrelevance reasoning in knowledge-based systems. Tech report STAN-CS-93-1482 (also KSL-93-58), Dept CS, Stanford Univ., CA. (Chapter 7).

[Miller et al., 1993] Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1993). *Five Papers on WordNet.* Prinston Univ., NJ. (http://www.cogsci.princeton.edu/∼wn/).

[Noy and Hafner, 1998] Noy, N. F. and Hafner, C. D. (1998). Representing scientific experiments: Implications for ontology design and knowledge sharing. In *AAAI-98*, pages 615–622.

[Schank, 1982] Schank, R. (1982). *Dynamic Memory.* Cambridge Univ. Press.