Semantic Modeling and Inference with Episodic Organization for Managing Personal Digital Traces

(Short Paper)

Varvara Kalokyri, Alexander Borgida, Amélie Marian, Daniela Vianna

Dept. of Computer Science, Rutgers University, New Brunswick, NJ 08903
{v.kalokyri,borgida,amelie,dvianna}@cs.rutgers.edu

Abstract Many individuals generate a flood of personal digital traces (e.g., emails, social media posts, web searches, calendars) as a byproduct of their daily activities. To facilitate querying and to support natural retrospective and prospective memory of these, a key problem is to integrate them in some sensible manner. For this purpose, based on research in the cognitive sciences, we propose a conceptual modeling language whose novel features include i) the super-properties "who, what, when, where, why, how" applied uniformly to both documents and autobiographic events; and ii) the ability to describe prototypical plans ("scripts") for common everyday events, which in fact generate personal digital documents as traces. The scripts and wh-questions support the hierarchical organization and abstraction of the original data, thus helping end-users query it. We illustrate the use of our language through examples, provide formal semantics, and present an algorithm to recognize script instances.

Keywords: personal digital traces, conceptual model, scripts, plan recognition

1 Introduction

Our modern lives produce digital traces consisting of "personal digital documents" (PDDs) resulting from sources such as email, messaging, calendars, social media posts, web searches, purchase histories, GPS location data, etc.[1]

Our goal is to use this information to help users supplement their retrospective and prospective memory, as envisioned in the "personal memex" of Vannevar Bush [4]. In addition to the problem of user interfaces for such systems, the key difficulty is integrating the heterogeneous and disparate kinds of PDDs.

Traditional research on Personal Information Management (PIM) has been "object centric", in the sense that the programs were intended to identify objects and represent semantic relationships between them, so that "finding" was supported by associative search . We believe that this is in part because of the field's origins lie in supporting office work, where finding files and other office objects was the standard use-case.

In contrast, a core feature of our work is the existence and exploitation of PDDs from a wider variety of sources. Each individual information source may have its own

[1] We immediately acknowledge the sensitive nature of this information, and the very important privacy issues that they raise.

natural (semi)structure (e.g., *from, to, date, subject, body* for an e-mail). What is needed is some way to integrate these "document schemas". One hypothesis embodied in our proposed conceptual model is that most information about a PDD can be fitted as answers to the questions *who, what, when, where, why* and *how* (the *w5h* questions), thereby providing a way of correlating the information on different kinds documents in a manner that is natural to humans. For example, the *who* property of an email include the sender and all the recipients, while the *who* property of a Facebook messenger/Google Hangouts conversation include the creator and the participants. Given this integration, one can provide relatively simple keyword search support along the "*w5h* dimensions".

A more significant novelty of our proposal, compared to traditional PIM, is motivated by the cognitive science literature (e.g., [15,16]), which shows that the intended use-cases are closely related to enhancing the user's *autobiographic memory*. This memory is centrally concerned with the *events* in one's life, which provide a narrative that connects the PDDs. For example, some emails concerning dinner, a confirmation of an OpenTable reservation, a Lyft receipt, and a credit-card payment, make much more sense as part of an episode of going out to dinner, *if* they have similar *when*, *where*, and *who* dimension values. Therefore, another hypothesis our work is that in developing a conceptual model for PDDs, one must make equal room for the modeling of events, both atomic actions and complex events. For the latter, we were inspired by the idea of *scripts* introduced by Schank and Abelson [13] for language understanding. These are stereotypical *plans* for common situations. Our current language for describing plans is based on Hierarchical Task Networks [7].

This paper illustrates the use of our proposed language through examples, provides a syntax and a formal semantics, and presents an algorithm to recognize script instances. It is a companion to the workshop paper [9] discussing some empirical results.

2 Related Work

We briefly mention here some of the many areas of related work.

The case for a unified logical data model for personal information has been made repeatedly in *PIM*, as has the use of ontologies (e.g., [11]) and semantic models like RDF(S) (e.g., [10]).

Since we are interested in representing (autobiographic) events and their instances, there is a vast literature on composite process and event representation spread across a wide variety of areas.

First, there are numerous formal process representation languages including program logics. Then there are the many graphical notations for describing real-world processes, such as Petri Nets, workflows and business process notations. These are all "prescriptive" in nature, while we are interested in "descriptive" formalisms that allow us to *recognize* script instances.

This leads us to several relatively closely related areas: *Activities of Daily Living, Ambient Intelligence, Behavior Recognition* and *LifeLogging*. A few relevant surveys are [2,12]. These areas separate two aspects: (i) the segmentation and recognition of

```
class DOCUMENT is a ENTITY {
   hasPart : set of ENTITY;
   who : set of PERSON
   what < hasPart: set of DOCUMENT;
   when : set of TIME;
   where : set of LOCATION;
   why : set of GOAL;      }

class SEND is a ACTION {
  sender < who: PERSON;
  recipients<who: set of PERSON;
  theme < what: DOCUMENT
  whenSent < when : TIME }
```

```
class EMAIL is a DOCUMENT {
 features:
  threadId : STRING;
 properties:
  from < who : PERSON;
  to < who : set of PERSON;
  ...
 actions
   send : SEND
   forward : FORWARD
 constraints
    from = send.sender ;
    send.whenSent < when;
 ... }
```

Figure 1: Specification of SEND and EMAIL classes

atomic actions from (continuous) sensor/video data; (ii) the recognition of complex events composed of atomic ones.

A significant number of the approaches for complex event recognition are founded on probabilistic techniques, which rely on Machine Learning of process schemas followed by probabilistic inference. While data sets for this are easily generated for sensors, they are much harder to obtain in our case because of privacy issues, and because (as we shall see) personalized variants predominate [9].

There is extensive literature in the field of AI on *plan recognition*. A snapshot of this appears in [8], and references to it. As with complex event recognition, probability-based approaches using learning are frequent. We seem to have more in common with approaches that use plan libraries for recognition. An interesting approach is the work of Geib et al [7], which is based on parsing hierarchical task networks (see Section 4), yet yields probabilistic results through "model counting".

The most important difference of our use cases from all the above approaches is that most of the digital traces we see are not part of any script, and a very large fraction of the plan steps in any particular instantiation of a script leave no trace ("missing actions")[2].

3 A conceptual model for entities and atomic actions

Real-world entities. We start from a standard object-centered conceptual modeling language, whose fundamental notions include *individuals* (e.g., Calvin) that are related by *binary properties* (e.g., hasFriends). Individuals are grouped as instances of *classes* (e.g., PERSON). Classes specify restrictions on the range of values that properties can take for their instances, and at least whether they are functional (**set of** indicates that

[2] The case study in [9] showed that 194 out of 316 episodes of eating out (61%) had *a single* PDD, corresponding to a single action in the plan associated with them.

there is no upper bound on the number of fillers). Classes can be specialized into subclasses (e.g., `RETIRED` is a subclass of `PERSON`), during which new properties may be added, or existing properties may be restricted. Most importantly, properties can also be specialized into sub-properties (e.g., `hasCloseFriends` is a sub-property of `hasFriends`). We will use the notation `hasCloseFriends < hasFriends` to indicate such specialization relationships between properties.

Actions. We have argued that a key part of the conceptual model for PIM are events. We focus here on modeling primitive/atomic events, which we call *actions*. The most important part of describing actions is presenting their participants, the roles they play and restrictions on them. We illustrate this in Figure 1, where the description of `SEND` includes properties for the important participants. So `sender < who : PERSON` is interpreted as saying that `sender` takes as value a `PERSON` instance, and is a sub-property of the `who` property.

Personal documents: We want to model PDDs in such a manner that *w5h* provide a unifying framework. For example, we will want to describe emails or reservations. Note however that there is no natural way to answer questions like "when?" or "who?" of such objects. (This is even more evident in the case of physical objects, like chairs say.) But if the object participates in an action, it can "derive" its *when* from the action. Thus one can ask when a message was sent, ... Therefore the model needs to express the natural properties of the PDD, connect to the actions involving it, and then assert the relation between their respective properties. One way to do this, inspired by our work on service description [3], is illustrated in Figure 1, where `EMAIL` is connected to the `SEND` action through the `send` property, and then **constraints** equate the `to` property of the email to the `send.sender` property path, which passes thru the `SEND` action. This equation can be used to *infer* one path value when the other one is known.

We can now re-state our original hypothesis: a large subset of the properties of PDDs as well as of the actions, can be usefully viewed as specializations of *w5h* (*who*, *what*, *where*, *when*, *why*, *how*), when these are viewed as properties themselves. The result is the principled integration of heterogeneous object schemas we suggested.

Semantics and Inference The above notation can be easily captured in UML, extended to support association hierarchies, which in turn can be translated to description logics (DLs) such as OWL[3]. (See [1] for an example translation.) DLs have precise formal semantics, and in fact languages such as OWL can express many more constraints, if needed. Several standard inferences are defined based on this semantics, and are supported by standard implementations of OWL, including *class inconsistency* (Is the specification of a class inconsistent, in the sense that it can never have any instances?) and *instance recognition* (Is an individual, with (partially) specified properties, necessarily an instance of some given class?).

One problematic feature of our language are **constraints**, called "complex role inclusions" in the DL literature. The general form we desire ($p_1 \sim q_1.q_2.\cdots.q_n$ where $\sim$ is $=, \sqsubseteq, \sqsupseteq$) is known to lead to undecidability of reasoning in most DLs. So such constraints can only be used to propagate information, using (epistemic) rules.

[3] `https://www.w3.org/TR/owl2-overview/`

4 Conceptual model for scripts

Our aim in using script instances is to *organize* PDDs, abstract out relevant information from them, and help humans access and make sense of episodes in their lives.

Therefore, relevant aspects of scripts include: (i) their goal (for purposes of human understanding; see [16,13]); (ii) summary information of the participants in the plan, and other descriptive properties, especially *w5h* aspects; (iii) the hierarchical decomposition into sub-scripts and primitive actions (together with restrictions on their ordering), which describe how the script plan achieves its goal.

Our system will start with a a library of common, everyday scripts. Since this plan library will not be used to construct plans, only to recognize instances that have been enacted, we can consider only *plan skeletons*, which ignore issues like pre- and post-conditions for performing actions. Also, since our application for organizing PDDs does not require perfect execution of all episodes, the plans we describe can be *stereotypical*.

We start here from Hierarchical Task Networks (HTNs) [5], which are a classical expressive notation in AI for planning and plan recognition. To describe the "body" of a plan, which determines the set of valid atomic action sequences, a non-atomic script/goal T can be refined in one of two ways:

$T := (\textbf{Or}\ S_1, ..., S_n)$ T can be accomplished by achieving *one* of $S_1, ..., S_n$;
$T := (\textbf{And}\ S_1, ..., S_n(P))$ T can be accomplished by achieving *all* of $S_1, ..., S_n$, subject to the precedence constraints $i \prec j$ in P, requiring S_i to end before the start of S_j.

The following frequently occurring patterns that can be expanded into the above **And** and **Or** constructs with the use of the **no_op** action:

sequencing	$S_1\ ;\ S_2$	$(\textbf{And}\ S_1, S_2\ (\{1 \prec 2\}))$
iteration	$(\textbf{Loop}\ S)$	$T := (\textbf{Or}\ (S\ ;\ T)\ ,\ \textbf{no\_op})$
optionality	$(\textbf{Optional}\ S)$	$(\textbf{Or}\ S, \textbf{no\_op})$

Future work concerns the addition of concurrency to the language.

Scripts also have properties, analogous to those of ordinary atomic actions and reminiscent of parameters/local variables in programs; and **constraints** relating these to the properties of their sub-scripts or actions mentioned in the body.

We illustrate these ideas by referring to Figure 2, which describes the `Eating_Out` ("going out to eat") script.

First, the values of `whoAttended`, `whenEating`, etc. describe and identify each script instance. Gathering the information into these properties provides the kind of higher level aggregation/organization of information in PDDs that we were looking for. Note that these properties are also organized along the *w5h* dimensions.

The body essentially describes the subgoals of the script plan. In this case, after an action initiating the idea of going out on this occasion, there are discussions about when, who, where (and hence what) to eat, which can be carried out in any order. Deciding when to eat in turn can be modeled by a script which shows exchanges of suggestions and discussions until agreement is reached.

As with regular classes, constraints like `AttendEatingOut.whatEaten = whatEaten`, can be asserted to propagate information between a script and its sub-scripts, if we assume each subscript is uniquely named.

```
 class Eating_Out  is a SCRIPT
locals:
 whoAttended < who: set of PERSON
 whereEating < where : EATERY
 whenEating < when : TIME
 whatEaten < what :  set of FOODS
 purpose < why : GOAL
body
  InitiateGoingOut ;
  (Loop  ( Or  DiscussWhenToEat,
    DiscussWhoWillEat,
    DiscussWhereToEat ) )
  (Optional  MakeRstReservation);
     AttendEatingOutEvent
  }  // end Eating_Out

class AttendEatingOut is a SCRIPT{
  body
   GetToEatery ;
   CheckIn ;
    (Or    (OrderFood ; BeServed),
        SelfServeFood ) ;
    Eat ; Pay ;  LeaveEatery     }
```

Figure 2: Definition of `Eating_Out` script and a sub-script

An instance of a script will have fillers for local properties, and an associated partially ordered set of sub-script and atomic action instances, which conform to the **body**. The major difference is that some participants and sub-scripts might be missing (because we may lack evidence for them in the form of PDDs).

Semantics. In the absence of conditions on states, the formal semantics of HTN interprets the above as a context-free-like grammar describing valid sequences of atomic tasks. An OR-decomposition corresponds to grammar rules $T \leftarrow S_1, T \leftarrow S_2, \ldots$. An AND-decomposition, when P is empty, corresponds to (exponentially many) rules for all permutations of $S_1, \ldots, S_n$. The above notation is powerful, because the task names can be used recursively, as in $S := (\textbf{And}\ (a, S, b), \{1 \prec 2, 2 \prec 3\})$.

5 Recognizing Script Instances from Documents

Let us review the context. We start with a model of the domain – classes for PDDs and scripts we are interested in. We then collect a large database of individual instances of PDD classes (the majority unrelated to any scripts). Our objective is to create episodes (instances of scripts) that the user was involved in, and relate them to documents. Algorithm 1 describes the steps for recognizing instances of script type $\mathscr{S}$.
We give further details of the algorithm steps, with reference to $\mathscr{S}$ = `Eating_Out` .

Step 1: Retrieving documents D: First, create a list L of "trigger words/phrases", whose occurrence indicates that a document has something to do with an instance of script class $\mathscr{S}$. To find these words, look for goal sub-script(s)/action(s), and identify verbs that indicate an occurrence of it. E.g., for `Eating_Out`, the goal is `AttendEating-Out`, and indicative verbs are "eat" or "eat out". Next, generate a list of synonyms and hyponyms based on these verbs. To make this process replicable, we have used standard sources of synonyms and hyponyms like WordNet. In addition, one must also consider the *w5h* participants of these events by using resources like FrameNet [6], and again generate synonyms and hyponyms. For example, "restaurant" is a discriminating *where* value of "eat", which should be included in the search term list. The final list includes

Algorithm 1 Algorithm for constructing instances of script class $\mathscr{S}$

1: D := documents indicating any potential instance of script class $\mathscr{S}$;
2: $Candidates := \emptyset$;
3: for all $d \in D$ do {
4: $Candidates$ += new instance c_d of script class $\mathscr{S}$, based on d;
5: rate the strength of evidence for c_d; }
6: repeat until no changes in $Candidates$ {
7: $MergeSet$:= { $d \in Candidates$ such that there is sufficient corroboration that they refer
8: to the same real-world event };
9: $Candidates := (Candidates - MergeSet) \cup \{d'$:=combine($MergeSet$)$\}$;
10: rate the strength of evidence for d';
11: use details of script $\mathscr{S}$ to look for additional documents that could be relevant to d'; }

"breakfast", "lunch", "dinner", and "restaurant", plus hyponyms. A set of documents D is retrieved by searching for these terms, and then preprocessing them by (i) explicating/disambiguating information (e.g. terms like "today" or "Tuesday" are made absolute dates), (ii) performing entity resolution, (iii) grouping certain kinds of documents (e.g., related email threads/tweets).

Steps 4&5: Creating initial script instances c_d. Each retrieved document d results in a candidate instance c_d of $\mathscr{S}$, with some of its (sub)properties filled based on the document's *w5h* properties. For example, a restaurant credit card charge provides evidence for the *attendEatingOut* sub-script, together with information on its *when*/*whereEatingOccurred*, and one *whoAttended* value (the cardholder). We then assign a score $Score(c_d)$ to c_d based on the strength of the evidence that d manifests. This strength is based on the document type (e.g. a restaurant credit card charge is *stronger* evidence than an email), the location of keywords (e.g. in the subject of an email rather than body), or of the originator (e.g., the user being the sender rather than recipient).

Steps 7−10: Growing script instances from MergeSet. In order to combine multiple sources of evidence for the same script instance, $\mathscr{S}$ needs to specify *"keys"*: a rating of how *w5h* (sub)properties help identify instances. For the `Eating_Out` case, important keys are *when*/*whereEatingOccured* and, to a lesser extent, *who*. Each key-property can be assessed for similarity (e.g. time difference for *when*). Once two instances of $\mathscr{S}$ are judged sufficiently similar, they become merge candidates, and are combined by unioning their property fillers. The score for the merged instance is $1 - \prod_{s \in MergeSet}(1 - Score(s))$. This formula is Hooper's rule [14] for combining probabilistic evidence.

6 Summary

This paper addressed the problem of managing a database of personal digital traces. It introduced a semantic modeling language for entities, also used to describe digital documents and related activities. The novel feature of this language is organizing many object properties into hierarchies with *w5h* questions at the top. These help organize and unify the many heterogeneous data . The language was extended to support the representation of scripts, which are stereotypical plans with a mereological hierarchical structure — an idea motivated by research in the cognitive sciences [13,16]. The

w5h organization was continued. Script instances connect the PDDs generated by actions into meaningful episodes, and extract from them relevant summary information, in order to reconstruct autobiographic memories.

Instance recognition for scripts, in contrast to plans and complex events, is complicated by several factors: 1) The evidence for the occurrence of atomic actions in the form of PDDs is highly uncertain (in principle, an email can describe *any* task in the world!). 2) Most PDDs we collect are unrelated to any of the everyday scripts we foresee having in the library, so these must be ignored. 3) Most of the steps in any instantiation of a script do not leave digital traces. For this reason, the paper proposed a novel heuristic algorithm for recognizing the instances of a script, which was based on retrieving PDDs that contained systematically chosen keywords, and merging candidate instances.

A small case study involving the `Eating_Out` script [9] gave instance recognition precision ranging from 0.32% to 0.75% per user[4], showing that there are major differences between subjects and how they generate PDDs.

References

1. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1), 70 – 118 (2005)
2. Bikakis, A., Patkos, T., Antoniou, G., Plexousakis, D.: A survey of semantics-based approaches for context reasoning in ambient intelligence. In: Euro. Conf. on Ambient Intelligence. pp. 14–23. (2007)
3. Borgida, A., Devanbu, P.T.: Adding more "DL" to IDL: towards more knowledgeable component inter-operability. In: Proc. ICSE'99. pp. 378–387 (1999)
4. Bush, V.: As we may think. The Atlantic Monthly July(1) (1945)
5. Erol, K., Hendler, J., Nau, D.S.: HTN planning: Complexity and expressivity. In: AAAI'94, pp. 1123–1128 (1994)
6. Fillmore, C.J., Johnson, C.R., Petruck, M.R.: Background to Framenet. International Journal of Lexicography 16(3), 235–250 (2003)
7. Geib, C.W., Goldman, R.P.: A probabilistic plan recognition algorithm based on plan tree grammars. Artificial Intelligence 173(11), 1101–1132 (2009)
8. Goldman, R.P., Geib, C.W., Kautz, H.A., Asfour, T.: Plan recognition (dagstuhl seminar 11141). Dagstuhl Reports 1(4), 1–22 (2011)
9. Kalokyri, V., Borgida, A., Marian, A., Vianna, D.: Integration and exploration of connected personal digital traces. In: Proc. of ExploreDB'17 Workshop. pp. 1–6. ACM (2017)
10. Karger, D., *et al*: Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data. In: Proc. CIDR'05, pp 13–26 (2005)
11. Katifori, V., Poggi, A., Scannapieco, M., Catarci, T., Ioannidis, Y.: OntoPIM: how to rely on a personal ontology for Personal Information Management. In ISWC'05 Wkshp. on The Semantic Desktop, pp. 258–262 (2005)
12. Rodríguez, N.D., Cuéllar, M.P., Lilius, J., Calvo-Flores, M.D.: A survey on ontologies for human behavior recognition. ACM Computing Surveys 46(4), 43 (2014)
13. Schank, R., Abelson, R.: Scripts, plans, and knowledge. In: IJCAI'75, pp.151–157. (1975)
14. Shafer, G.: The combination of evidence. Int. J. of Intelligent Systems 1(3), 155–179 (1986)
15. Tulving, E.: Episodic memory: From mind to brain. Annual Rev. of Psych. 53, 1–25 (2002)
16. Williams, H.L., Conway, M.A., Cohen, G.: Autobiographical memory. In: Cohen, G., Conway, M.A. (eds.) Memory in The Real World, chap. 3, pp. 21–90. Psychology Press (2008)

[4] Low-end scores were due to factors such as absence of NLP and a couple sharing credit cards.