Pini Language and PiniTree Ontology Editor: Annotation and Verbalisation for Atomised Journalism

Guntis Barzdins[1,2,3], Didzis Gosko[1,3], Karlis Cerans[1], Oskars F.Barzdins[3], Arturs Znotins[1], Paulis F.Barzdins[1], Normunds Gruzitis[1], Mikus Grasmanis[2], Janis Barzdins[1], Uldis Lavrinovics[2], Sinty K.Mayer[3], Intars Students[2], Edgars Celms[1], Arturs Sprogis[1], Gunta Nespore-Berzkalne[1], Peteris Paikens[1]

[1] Institute of Mathematics and Computer Science, University of Latvia, Riga, Latvia
{guntis.barzdins, didzis.gosko, karlis cerans}@lumii.lv
[2] LETA, Marijas street 2, Riga, Latvia
[3] PiniTree, EU

Abstract. We present a new ontology language Pini and the PiniTree ontology editor supporting it. Despite Pini language bearing lot of similarities with RDF, UML class diagrams, Property Graphs and their frontends like Google Knowledge Graph and Protégé, it is a more expressive language enabling FrameNet-style natural language annotation for Atomised journalism use case.

Keywords: ontology languages and editors, natural language processing

1 Introduction

We address the problem of describing real-life situations (facts about past, Atomised news [1]) in a formal language close to natural language and easily understandable by the domain experts and end users.

The primary construct necessary in such descriptions is a subject-predicate-object relation, with a possibility to add secondary property-value pairs to the relation.

Basic RDF [3] modelling of this would involve creating a resource for the relation fact, thus bringing the data model far from the linguistic representation.

UML [2], Property Graph [4] and RDF* [8] notations allow for direct property ascriptions to the links (statements in the case of RDF*) in the data model, however, with certain limitations:

- UML, by its design philosophy, allows only one link for a link type (an association class) between an object pair (cf. [2], p. 438),
- Property Graphs allow only scalar values as link attributes, and
- RDF* aggregates all annotations on a subject-predicate-object triple together, thus also excluding several same-predicate links between the same objects.

This paper proposes a new *Pini language* with sentences and paragraphs close to the natural language and a simple data model resembling Wikidata predecessor graphd [9] that do not suffer from the abovementioned data model limitations.

Requirements for the Pini language come from the real-world text modelling experience [7] with the Berkeley FrameNet [6]; the novelty here is to define two frame elements for each considered frame as the subject and object; the other frame elements become secondary properties of the main subject-frame-object relation.

The Pini language is implemented in the *PiniTree* editor (available at *pinitree.com*) tested on LETA News Agency and BBC use-cases – CV extraction from a news archive [7], and Atomised journalism [1] – and demonstrated in this paper.

2 Pini Language

The *Pini language* can represent knowledge about a wide range of domains. The basic element of the Pini language is a *Pini entity,* which represents a discrete object with well-defined boundaries and identity in the physical or imagined world [2, p.442].

The Pini entity has three attributes: *type*, *lexical name* and *GUID* represented as:

[(type) LexicalName (GUID)]

The *type* denotes a class to which the Pini entity belongs, such as *person, organization, place, relation*. It helps disambiguating the lexical name of the Pini entity, as "Paris" could be either a *location* or a *person* name.

The *lexical name* is a canonical name by which the Pini entity could be referenced in natural language, for example *Peter, Nokia, Finland, located_in*.

The *GUID* values are globally unique random identifiers (unlike the globally coordinated URIs for resource identification) by which Pini entity is uniquely referenced.

Pini entities are subdivided into *item entities* and *link entities*. Item entities exist on their own and they represent objects like *Finland* and *Nokia*. Link entities represent a concrete relationship between two other Pini entities, as, e.g., in the Pini triples:

<[(org) ***Nokia*** *(...)], [(relation)* ***located_in*** *(45af23…)], [(place)* ***Finland*** *(...)]>*

<[(org) ***Ericsson*** *(...)], [(relation)* ***located_in*** *(7e53b4…)], [(place)* ***Sweden*** *(...)>*

Note that although the relation lexical name "*located_in*" is the same in both examples, these relations are different Pini entities since their GUIDs are different.

The *Pini triple* is an ordered list of three Pini entities referred to as <*subject, predicate, object*> respectively, where the predicate must be a link entity and the object must be an item entity (the subject can be either an item entity or a link entity).

The *Pini ontology* is a set of Pini triples, where every link entity appears as a predicate in exactly one Pini triple. A Pini ontology can be visualized as a graph in which all Pini entities with the same GUID are collapsed in the same node (cf. Fig.1).

Pini sentence is a fragment of the Pini ontology starting with a single Pini triple

<[subject-entity], [predicate-entity], [object-entity]>

and it includes all secondary Pini-triples in which the above *predicate-entity* shows up as the subject entity. With well-chosen lexical names such Pini sentences easily map to a natural language sentence such as "*Steve Jobs wife was Laurene Powell and they married in 1991 at Yosemite National Park."* (see Fig.2).

The *Pini paragraph* is a set of all Pini sentences sharing the same focus-entity (see Pini paragraph for the focus-entity *Steve Jobs* in Fig.2).

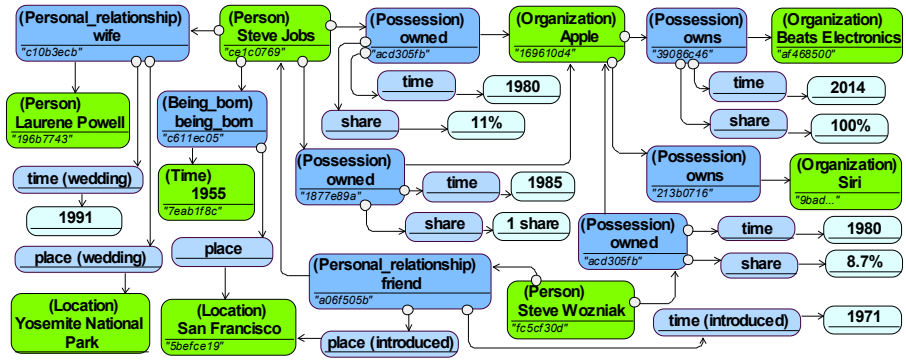

Fig. 1. Example Pini ontology. Note two separate triples <*Steve Jobs, owned, Apple*> with different attributes.

Pini literal is syntactic sugar for self-describing item entities appearing only in the object of some Pini triple, like "184cm" in:

<[(person) ***Peter*** *(...)], [(relation),* ***height****, (4bv15f...)], [(literal)* ***184cm*** *(...)]>*

Pini literals can be included in the lexical name of the relation to omit the object:

<[(person) ***Peter*** *(...)], [(relation),* ***height:184cm****, (4bv15f...)], - >*

Pini literals are depicted light blue in Fig.1 and Fig.2.

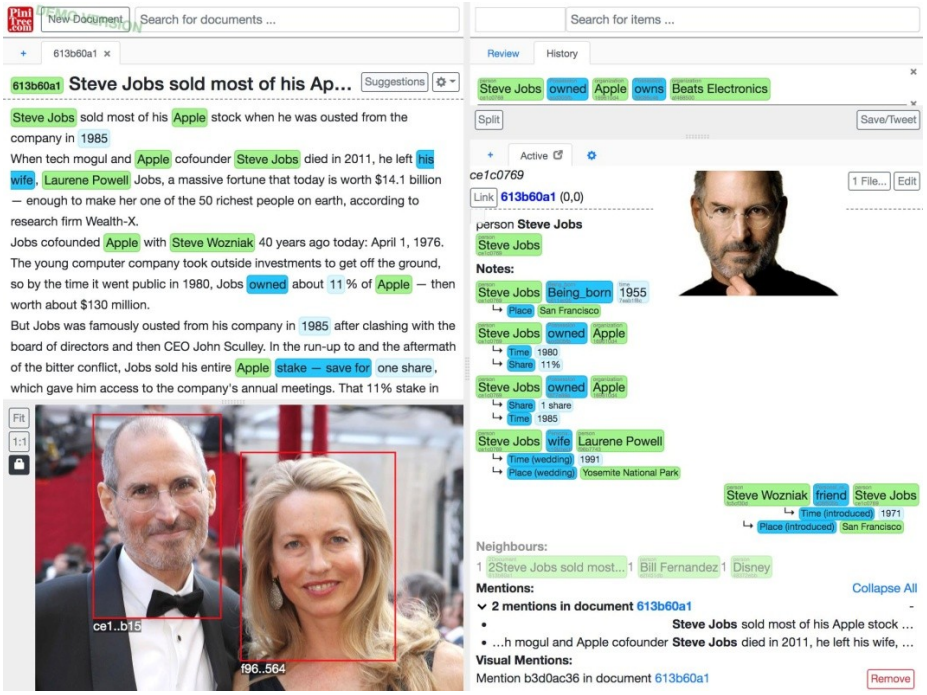

Fig. 2. Pini ontology fragment in the PiniTree ontology editor.

3 PiniTree Ontology Editor

PiniTree is an editor implementing the Pini language for the Atomised journalism use-case. As shown in Fig.2 it has two distinct panes: the right pane is the Pini ontology editor and the left pane is the Atomised journalism workbench. The left pane displays *Pini documents* (text with images) resembling Wikipedia articles while the right pane allows navigating and editing the ontology resembling DBpedia. But unlike DBpedia, which integrates with Wikipedia only on the article level, PiniTree editor integrates the Pini ontology and Pini documents on the word and sentence level by means of Pini mentions.

Pini mention is a feature of the PiniTree editor enabling referencing Pini document segments from Pini entities as a source of attribution – a grounding feature missing e.g. in Google Knowledge Graph [5]. Pini documents are accumulated as read-only objects in the PiniTree editor and are assigned a unique Pini item (GUID 613b60a1… in Fig.2) holding the document metadata. Document GUIDs along with the segment offset serve as the Pini mention target. Besides manual disambiguated entity mention annotation assisted through the entity spelling *aliases*, the PiniTree editor also suggests entity mentions similarly to Google search using neural contextual word embeddings and neural face recognition. Link entities are suggested based on frequently co-occurring neighbouring item entities (e.g. "Disney" in Fig.2). GUIDs and aliases rather than Wikipedia page names stimulate broad synset use as Pini entities.

Pini ontology editor in the right pane assumes that human perception of the Pini ontology naturally is based on sequential navigation through the neighbouring Pini paragraphs. A Pini paragraph is the unit of information one can perceive simultaneously as the episodic memory. Navigating through Pini paragraphs forms a linear history of focus-entities in the short-term memory to be re-accessed easily.

The Pini ontology graph in Fig.1 can't deliver this experience as in real applications it may become very large and incomprehensible. Instead the PiniTree editor represents Pini paragraphs as illustrated in Fig.2. It supports navigation between the Pini entities by clicking on them like in a web browser; unlike the browser "back" button the entire browsing history is accessible at the top-right pane. The browsing history often resembles a short story and can be saved in a new Pini document as the blue-print for the Atomised journalism output.

4 Adding Structure to the Pini Language

The Pini ontology in Fig.1 and Fig.2 is easily understandable because it has clear structure. There are many ways to structure a Pini ontology – e.g., an alternative Pini structure equivalent to Google Knowledge Graph [5] or Wikipedia infoboxes is illustrated in Fig.4. But for journalistic use-cases we need more granular n-ary relations better captured by the Berkeley FrameNet (see Frame example in Fig.3a) requiring full Pini language expressivity for secondary attributes (see *SanFrancisco* in Fig.2). The visualisation and editing in the PiniTree editor is universal and supports infobox and FrameNet structuring, as well as unstructured "linked data" approaches.

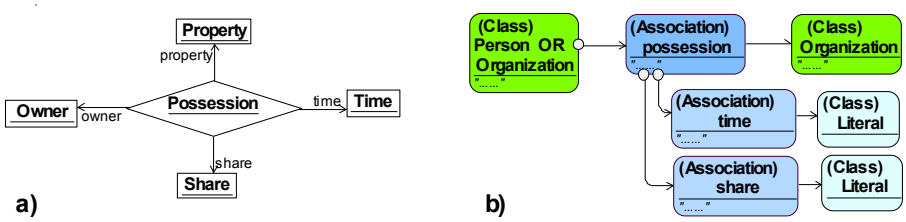

Fig. 3. Frames in PiniTree: FrameNet notation (a) and Pini frame notation (b).

We define a *Pini frame* to be a FrameNet frame in which two frame-elements are identified as a binary subject-predicate-object *core association*, where predicate type is the frame-name. Other frame-elements attach to the predicate of the core association as subject – these we will regard as secondary roles (see Fig.3b). Pini sentences are instances of such Pini frames. Formally, a Pini frame is a Pini ontology with the meta-types *Class* and *Association*. Pini frames constitute the terminological part of the Pini ontology and are stored in the separate *Ontology* meta-type used by PiniTree editor to soft-constrain the regular Pini entity types.

Fig. 4. PiniTree view of the Google Knowledge Graph or WikiPedia infobox data.

The example ontology in Fig.1 and Fig.2 uses Pini frames derived from the corresponding FrameNet frames: *Possession, Personal_relationship*, and *Being_born*. The core association among the roles constituting the frame is identified by studying the syntactic realization and valence patterns of the frame, which are part of the FrameNet dataset derived from large manually annotated text corpora.

5 Discussion and Conclusions

The three FrameNet frames used in above example are insufficient for the LETA and BBC use-cases. In the LETA use-case [7] we found that seven FrameNet frames *Being_born, Death, Personal_relationship, Education_teaching, Being_employed, Membership, Possession* are sufficient for their use-case of extracting politician CVs from their news archive. On top of these 7 frames the BBC Atomised journalism use-case [1] also requires *Participation* and *Statement* frames, as daily news typically revolve around events and their participants along with any notable statements made by political influencers.

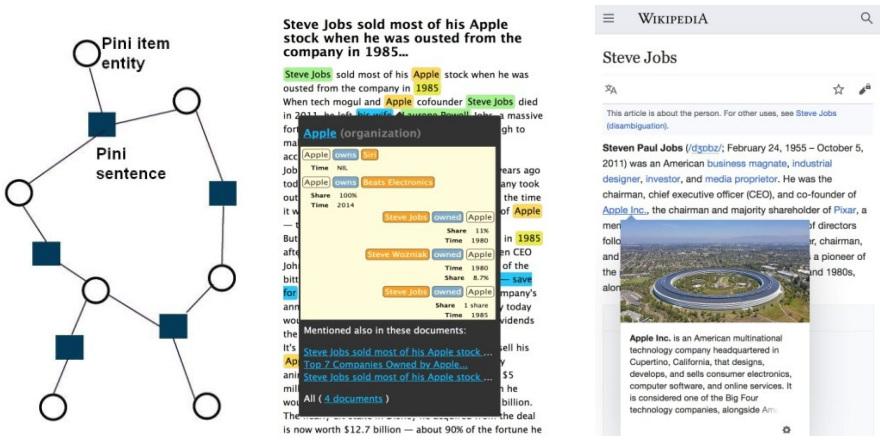

Fig. 5. Pini WoIB structure and end user view resembling Wikipedia page preview popup.

The end user Web of infoboxes (WoIB) view in Fig.5 illustrates the Pini enriched web page with navigation popup resembling a Wikipedia page preview with the Pini infobox and mentions. We are optimistic that the described approach can be further extended with neural question answering [10] and reasoning.

Acknowledgments. The research leading to these results has received funding also from the ERDF project 1.1.1.1/18/A/045 at IMCS, University of Latvia, and from the project "Competence Centre of Information and Communication Technologies" of EU Structural funds, No. 1.2.1.1/18/A/003, Research No. 2.4 "Platform for the semantically structured information extraction from the massive Latvian news archive".

References

1. Rhianne, J. and Bronwyn Jones, J.: Atomising the News: The (In)Flexibility of Structured Journalism. Digital Journalism, Vol.7(8), 1157-1179 (2019).
2. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. 2nd edn. Addison-Wesley (2005).
3. Resource Description Framework (RDF), http://www.w3.org/RDF, last accessed 2020/05/05.
4. Robinson, I., Webber, J., Eifrem, E.: Graph Databases. O'Reilly Media, (2013).
5. Google Knowledge Graph, https://developers.google.com/knowledge-graph, last accessed 2020/05/05
6. Fillmore, C.J., Johnson, C.R., Petruck, M.R.L.: Background to FrameNet. International Journal of Lexicography, Vol.16, 235-250 (2003)
7. Barzdins, G., Gosko, D., Rituma, L., Paikens, P.: Using C5.0 and Exhaustive Search for Boosting Frame-Semantic Parsing Accuracy. In: LREC2014, pp. 4476-4482 (2014).
8. Hartig, O.: Reconciliation of RDF* and Property Graphs. In: arXiv:1409.3288, (2014).
9. FreeBase graphd Repository, https://github.com/google/graphd, last accessed 2020/05/05
10. Dhingra, B., et al.: Differentiable Reasoning over a Virtual Knowledge Base. ICLR (2020)