The Monoparser FSM as a Flat-to-Graph Transpiler

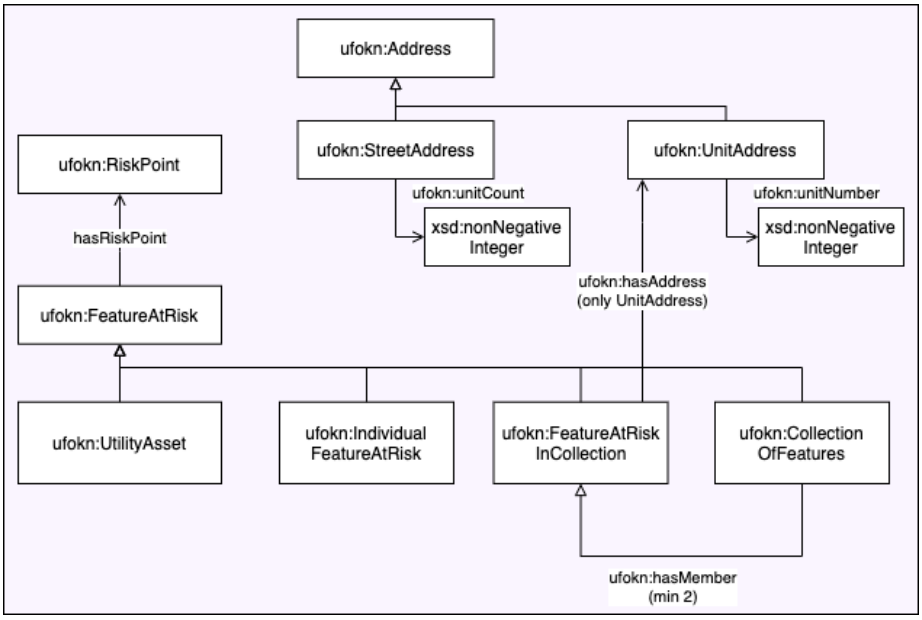
Sean McWillie (UFOKN) sean@seanmcwillie.com

Converting flat storage formats such as parquet or XML into graph data statements cannot be done using a general purpose algorithm. That is due to the semantic nature of the data. To facilitate machine-assisted conversion of parquet files, a storage format for flat columnar data, into RDF statements (states following a Subject-Object-Predict triple form).

In order to convert parquet files into graphs the following standard pipeline is used:
Parquet file > Pandas dataframe > List comprehension mapping and routing values according to their status

This follows the general convention of converting a domain-specific originating data format into a standard format (Pandas dataframe, CSV) and then converting that standard data format into the target format.

The graph schema for these single and multi-unit Features as follows[1]:



Both the originating target data formats require linkage to each other, and as such can be modeled as a wiring diagram, which can be implemented as a desktop GUI-based application that exports originating values with target values (allowing new features to be added and procedural refactoring). This wiring diagram, shown below, can be implemented as a type of finite state machine called a monoparser (which treats features as a whole rather than splitting them up into nested loop statements and incurring amortized runtime issues). Because the list processing is linear, it is trivial to split up flat datasets and run the monoparser. More importantly, the monoparser is built around treating all rows in the flat data set as single entries, avoiding nested loops (for loop, or comprehensions).

This transpiler takes an input in a flat format and extracts embedded information, using a finite state machine that strides down the list of incoming data, instantiating the columnar

[1] From previous work by Torsten Hahmann and Sean McWillie (2021, 2022).

values as graph statements that are linked to each other according to a graph schema. The wiring diagram can be updated using basic UNIX commands such as sed, or any modern text editor. The code itself forms an interstitial format that combines Python code with RDF statements and as such can be used for templating and dynamic updating.

Columnar Data			Graph data	
	source	osm-ms-oa	Feature Property	
	buildID	UFOKN-dq0dtevjev	NamedIndividual	use for URIs
	ms_id	dq0dtevjev	Feature Property	
	osm_id	935890693	Feature Property	
	osm_model	way	not in ontology	omit
	KEY	building	Feature Property	
	VALUE	yes	Feature Property	
	description		not in ontology	omit
	oa_id	['00027c4330b507c6']	Feature Property	linkage to Address data properies?
	unit_count	1	not in ontology	omit
	number	['102']	Address Property	
	street	NORTH CHANNEL DRIVE	Address Property	
	units	['']	Address Property	omit
	city		Address Property	meta-data?
	district		Address Property	can get this using zipcode
	postcode	28480	Address Property	can get this using zipcode
	geometry	POINT (1654323 1397477)	RiskPoint Object	
	area	245.0533604	not in ontology	

This embedded information[2] takes the form of either:

a) atomic values (such as a street name -- a building generally has one and only one canonical street name),
b) atomic values in a list (in particular, subunits of a multi-unit building, eg. the rooms in an office), or
c) key-value pairs (the type of feature along with a more specific subcategory, such as Amenity: Place of Worship). Remains unimplemented, pending clarification from source data custodians.

Case a) is trivial -- the graph statements can be grafted over columnar values using basic string manipulation and joins.

For case b), this requires nested processing (in quadratic time), or a way to map lists as flattened information to take place as part of the flattened processing stream (linear but with functional overhead), which can be accomplished with anonymous functions as the parent feature's information needs to be linked to the subunit features' graph statements.

Case (c) is not yet implemented at this time.

---

[2] Information from LHS of chart from UFOKN Tech Team members Mike Johnson, Adam Shepherd and Douglas Fils (2021).