A Better Model

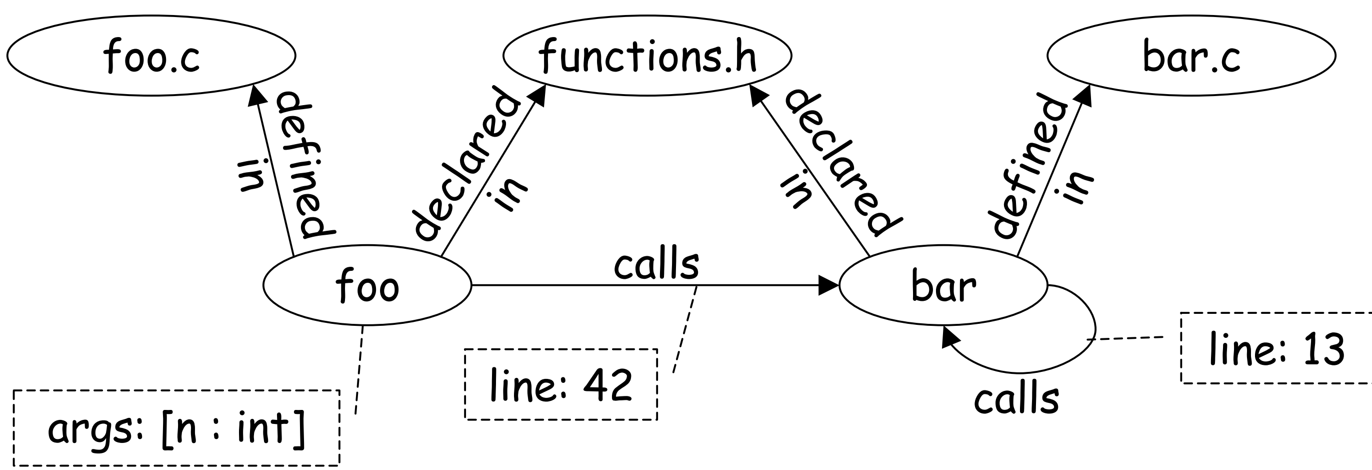
Piotr Kaminski

<pkaminsk@who.net>

Semantic Nets

- An idea from the field of Artificial Intelligence
  - represent information as a collection of concepts
  - meaning arises from the relationships between the concepts
  - applicable to many domains

- Usually represented using graphs:



Roadmap

- Talk concentrates on the abstract semantic net meta-model
  - no specific information domain
  - not conversion of data to a semantic net
  - not much about visualizing the semantic net
  - very little about a possible implementation

1. Problems and solutions
2. Types (with a diagrammed example)
3. Order and sequencing
4. Queries and inferences

Problems

- Why are attributes different from relationships?
  - requires two different access mechanisms
  - must know how each property is represented for queries
  - attributes may have further internal structure that ought to be captured in the graph
- Why are edge labels not a normal attribute?
  - requires special constructs when writing queries
  - can't represent the concept of edge type within the graph
    - trying to do so can lead to infinite regress
- Why are all relationships binary?
  - imagine a relational database that only allows two-column tables
  - could work around this by indirecting all edges through an intermediate vertex

Foundations

- Use a *hypergraph* :
  - each edge may connect any number of vertices
  - same theoretical foundation as graphs

- Vertices are not attributed
  - represent attributes by binary relationships between the subject and the value
  - requires special “atom” vertices to represent the actual values

- Edges are collections of vertices
  - cardinality may be greater than 2
  - each edge is also automatically a vertex (a *collector* or *relator*)
  - allow edges to be members of themselves

- Call vertices *entities* and edges *relationships*

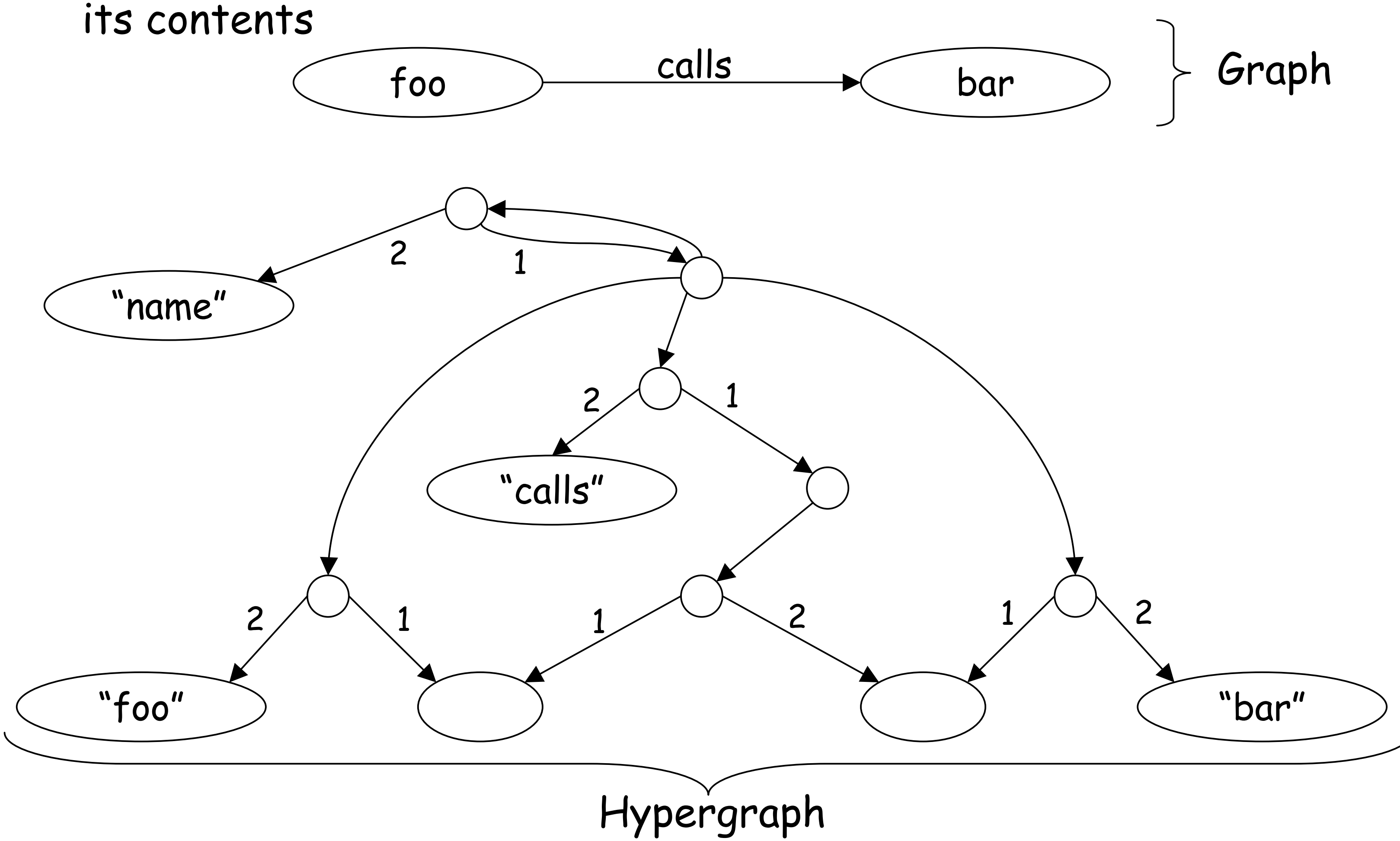
Types

- Avoid strict typing
  - keep model flexible for experimentation and integration
  - allow optional validators that indicate any problems

- Entities (including relationships) are collected into types
  - a type is a relationship that collects all vertices that are instances of the same concept
  - an entity type might be “function” or “file”
  - a relationship type might be “calls” or “line number” or “name”
  - another type is “relationship type”
    - note that this type is a member of itself

Example Diagram

- Each entity and relationship is a vertex
- A relationship has unlabeled binary directed edges from itself to its contents



Don’t Panic!

- It looks complex, but…
  - only the initial setup requires lots of extra vertices
  - long term, the # of vertices is O(n) relative to a plain graph
  - an implementation is not required to realize a vertex until it’s needed by the user
    - most vertices will be virtual for their entire lifespan

- How to present these hypergraphs to the user?
  - abstract away from the physical structure
  - similar to usual Rigi and SHriMP visuals
  - n-ary edges represented by either tentacles or grouping
  - potential issues with relationships that contain themselves

Ordering

- Order is a basic concept, viz:
  - the order of a collection of function calls could be important
  - the meaning assigned to each member of a “name” relationship is different

- A relationship has some basic properties that affect its contents:

Distinct elements?	Order	Common name
yes	none	set
yes	partial	poset
yes	total	ordered set
no	none	bag
no	partial	pobag ?
no	total	list

- Also, a list with a fixed size is a *relator*

Sequencing

- Sequence is also a basic concept, distinct from order
  - some infinite sets have order but no sequence (e.g. the rationals)
  - sequence does imply order, though

- Sequencing is more than a series of binary relationships
  - allows access to entity at any given index
  - changing the sequence of a linked list is tricky
    - especially if only certain operations are allowed by a given relationship (e.g. exchange elements, move elements)

- However, if a collection of elements is already sequenced using binary relationships, we can take advantage of it

Queries

- A query selects a subset of an input relationship to produce a new output relationship
  - the result changes dynamically along with changes in the input
  - the output relationship is read-only
  - the output relationship can be used as input for another query
- How?
  - make a new language? rule-based?
  - how to describe the ordering/sequencing of the result?
  - what's a useful interpretation of transitive closure in a hypergraph?

Inferences

- There are always implicit entities and relationships in a graph that can be inferred from existing information, e.g.
  - aggregation: combine relationships along some hierarchy to allow for high-level overview of relationships
  - factoring: split relators into a bunch of binary relationships, and vice-versa
  - reification: make the containment relation explicit (as in the raw diagrams), etc.
  - any other derived semantics…

- Inferred entities should be indistinguishable from original ones
  - can be visualized alongside the original entities
  - can be related to other entities, whether inferred or original

- Inferences are similar to queries, but create new entities
  (instead of just filtering and regrouping existing ones)

Virtualization

- We don't necessarily want to represent all results of an inference explicitly:
  - the number of inferred entities could be big (even infinite)
  - most of them might never be needed
  - often, we just want to know if an existing entity participates in some inferred relationship
- Hence:
  - let user specify explicitly which inferences to apply where
  - apply inference rules lazily
  - cache inference results
  - destroy inferred entities if unmodified and no longer needed
- All (de)virtualization is performed transparently!

Conclusion

- Advantages of this hypergraph meta-model:
  - uniform: simplified access from user code and reuse of visualization code at all meta-levels
  - abstract: virtualization allows complex inferences and infinite sets without destroying uniformity
  - complete: can represent all levels of meta-concepts within the graph itself
  - flexible: can be used to integrate many other models
  - rich: offers many opportunities for optimization

- What's next?
  - ironing out queries and inferences
  - specifying dimensions
  - implementing a proof of concept or two.