Jason C. Stockwell
openpatterns.net
March 17, 2016
v0.1.1

Theory and API of the Ents Database Concept

This paper describes the idea of the Ents database, designed to hold specific types of factual information for smart editing and analyzing. These ideas are only preliminary and there are many improvements to be made, big and small.

Ent is short for entity.

Axioms

- An ent represents any number of things or ideas.

- Two ents are *connected* if one is a subset of the other.

Basic Terminology

Uppercase letters A, B and C all represent different hypothetical ents.

If B represents a certain classification of things, and A represents exactly those things and more, then B is a *subset* of A. For instance, an ent representing mammals is a subset of one representing animals because all mammals are animals, but there are animals which are not mammals.

If A and B are connected and B is a subset of A, then A is called a generalization of B, or "*gen*" for short, and B is a *sub* of A, short for subset. All ents which are subsets of A are A's *subs*, and all ents which A is a subset of are A's *gens*.

If A is a gen of B, they are *directly connected* if A's subs don't overlap B's gens. This means there is not a third ent C which is a sub of A and a gen of B. A *direct gen* or *direct sub* is a gen or sub that is directly connected.

Two ents whose subs can not overlap are said to be *exclusive*. Just because two ents have the same gen does not mean they can't overlap. For instance, directly under animals we could have mammals and land animals, which will surely differ in their subs, but they also overlap. However, directly under animals we could have lizards and mammals, which would then be exclusive. The information of whether ents are exclusive or not can be used extensively.

Logical Derivations

Logically, every Ents database may as well contain an ent which represents "everything". We will call it *root* in reference to the hierarchical tree. Any unsorted ents will be placed by default directly under root.

Every other ent must logically be a subset of root.

Discussion of an entity that represents “nothing” is irrelevant.

An ent's gens and subs can not overlap in any way, otherwise there is a logical inconsistency.

If A is exclusive to B, A is also exclusive to all subs of B.

As a Library

I believe the concept of the Ents database has many uses. Therefore it makes sense for it to be a standalone module for incorporation in other projects. Thus it should be designed with generality. This includes developing it for different platforms and in different programming languages.

By being a library, we make it easier for anyone to incorporate it in their projects. It should also have the ability to create standardized databases for use in multiple software applications developed independently.

I just wanted to convey this intention of inclusiveness so people are aware.

API

This API is extremely preliminary and is just a basic list of essential functions. It is intended only as a starting place and example of how to approach the Ents concept programmatically.

An operation on the database which maintains logical consistency is a *legal* operation. By only performing legal operations, we may prevent the user from making mistakes, and perhaps bring any previously made mistakes to the user's attention. We can't automatically find all mistakes, but we can at least find the ones which break the logic already within the tree.

One area of discussion will be on how to handle connections between ents. If A is a gen of B and B is a gen of C, then is it necessary to keep the connection between A and C, or is that superfluous? Is the drawback to storing this connection worth the potential computational advantage. This becomes more apparent in large databases.

An important action to plan for is organizing. If a user adds a new ent, it could first be made a direct sub of root. Then, the user may be strategically asked if that ent is a sub of any other ents, and so on and so forth, thus exhausting its organization in the existing tree.

For instance, a user wants to add an ent representing humans.

A series of questions may be “plant or animal”, “mammal or reptile”, “ape or marsupial”, etc. This is opposed to presenting the user with perhaps 50 choices to sift through at once. By utilizing ent exclusivity we may derive extra information and avoid illogical questions like if humans live under water.

The system could choose which questions to ask first in order to increase efficiency. One could refrain from asking about rare properties at first which may be later deduced automatically based on exclusivity. Further more, this could be optimized somewhat by weighing questions and using a path of least action approach, or rather a path of least questions.

For example, if one is adding a living organism to the database, a logical first question may be “What kingdom is this apart of?” not “Is this an orange ape?”

If there are millions of ents in a database, then storing all of their connections may be difficult. But if only direct connections are stored, we will need to calculate gens and subs on demand for certain situations. It remains to be seen how these factors will interact in practice.

The Ent software object (style of Java)

Object Ent X represents an ent. It should have a unique identifier and a name and description. It would make sense that names are unique as well.

X references it's direct gens and subs, as well as the ents which it is directly exclusive to. This sufficiently describes how it relates to other ents.

All Ent objects will be stored in hash maps to quickly look them up via IDs or names.

int id – Unique unsigned identifier to differentiate X from other ents. Root's id may well be 0.

int[] dGen, dSub – Array of ids of X's *direct* gens and subs.

int[] exc – IDs of ents directly exclusive to X.

API for manipulating ents in the database

These methods/functions allow for valid transformations on the database elements that maintain logical consistency.

Basic getter, setters, etc. are not listed.

Importantly, each ent must be reachable by starting at root and going down the tree via direct subs. Thus, any operation on the tree should not create an unreachable ent.

It may be advantageous to keep running arrays of an ent's entire list of gens and subs, not just direct ones, in order to increase computing speed, but that will not be explored here.

Methods/Functions

newEnt(genID) – Creates a new Ent object with a new unique id and sets the ent with id genID as a direct gen, adding it to dGen[]. If there is no argument, set it's direct gen root. One may wish to create an ent as a sub to an existing ent.

Object connect(int G, int S) – Connects two ents G and S as direct gen and sub respectively. For this to be a valid operation, the system must verify that none of G's gens overlap with S's subs. If rejected, the user should be informed what ents overlap because there is a mistake. Could return an object with information as to the success of the connection.

Object removeEnt(int ent) – Removes an ent from the database completely. If the ent has no subs, this isn't much of a problem. If it does have subs, they may become unreachable. If none of its direct subs become unreachable, it could be safely removed. The user may however want its subs to be connected to its gens, maintaining logic but taking out the middle-man. The user may also want to remove the ent and all of it's subs. Return element could indicate problems.

Object disconnect(int G, int S) – This removes a connection between two ents and abstractly represents a property being removed from an ent. S must not be left without a direct gen, because it will then be unreachable via direct movement within the tree. Return element would notify of problems.

Object validate() - This method would validate the logical consistency of all ent connections. If a logical inconsistency is located, then it must be dealt with. By limiting operations on the database to one which maintain logical consistency, this shouldn't occur, but it may be useful for detecting bugs. Return element could describe error.

Further Thought

There are many ways to represent a hierarchy. The aforementioned methods have each element keep references to its direct connections. This allows each element to be reached, but may make some calculations inefficient, like generating a list of all subs of a ent.

One alternative or supplemental strategy could utilize prime numbers. Each ent would have a unique positive integer which mathematically relates to the integer of other ents. If A were a gen of B, then B's number would be divisible by A's. The simplest example would be of root, which would have a number of 1. Each ent is a sub of root, and each integer is divisible by 1. This could make verifying distant gens and subs as easy as a few numerical calculations. However, this may be limited due to numbers becoming too large for calculation, it depends on how this scales.

Other analysis on the hierarchy could then be implemented mathematically and perhaps exploit advances in number theory. Using both systems side by side one could use the fastest method for the task at hand if memory size were not an issue.

Todo

Discuss theory of generating questions to organize and classify ents.

Work through how to accommodate words and syntaxes of a language within the logical hierarchy. Syntaxes are specific orders of words which represent something. If there are 3 words in a syntax, each word may be restricted to a number of word types. These word types would be ents in themselves. Could this be done significantly within Ents itself or will it require an additional logical property to be added?

A problem involves how to taxonomically sort the specific spellings of a word. For instance, two words with the same spelling represent different things. The existing basic Ents database may be insufficient to provide all of the detail needed, but it could organize things which to be further augmented and supplemented by the system actually doing the language analysis.

After a working version is created, various usability improvements will be needed to deal with common problems of modifying the database.