Hoovers Narrative Generator Demonstration Prototype

—Marc LeBrun 8/3/2010

Overview

Presenting database information in the form of a natural language narrative text can be a more effective and user-friendly interface than just listing raw facts and figures.

To demonstrate how this concept might work in practice a prototype was developed that accesses a Hoovers company information triple-store database for input, then automatically generates a report in narrative form presenting this data as output.

This naturally provided a testbed to explore how semantic technology might be exploited in implementing such a system, and to highlight opportunities for delivering new features of value to Hoovers customers (such as reporting additional facts of potential interest inferred from, but not explicitly stored in, the underlying raw data).

Moreover, although this project was scoped primarily to just demo the functionality, it was designed and implemented in such as way that it can be leveraged to help develop a more "production" grade system as a later phase if desired.

Implementation Outline

The system is built around *templates*, which are "fill in the blank" or "Mad Libs" sentence structures that consist of a sequence of fixed text strings and placeholders for variable information to be supplied later. The current implementation uses templates that represent sentences that might appear in a narrative. Some example templates are

```
("A leading vendor of " :line_of_business " is " :company_name)

("Headquartered in " :primary_city ", " :primary_state ", the company employs " :total_employees)
```

In generating a report the system *renders* a template by replacing the variable placeholders with the specific information pertaining to the subject company to produce readable sentences:

```
A leading vendor of software systems is Franz Inc.

Headquartered in Oakland, CA, the company employs 40.
```

(Note that templates can be readily localized for different languages, leveraging the required translation effort since a single template can be reused in many reports.)

To generate a full report the system first builds up a *narrative* consisting of a sequence of templates, which it then renders. The system selects the specific templates to be used for given a narrative from a *pool* of templates. Currently the demo implementation is initialized with a pool containing roughly a couple dozen templates (including the two examples above—see below for full list). However the pool size can be increased as desired simply by authoring more templates.

Of course a notable property of reports written by human editors is that the same information can be expressed and presented in a variety of ways, rather than in a single format in a rigid order.

In order to simulate this desirable feature, the system uses some elementary but effective AI techniques to generate a narrative by using a randomized backtracking search heuristic. Rerunning the generator typically outputs different narratives.

Here are two consecutive runs on one company in the database (DUNS # 440040):

```
-------- {4400040} --------
Based in Hampton Cove, AL, Burleson Pool Company, Incorporated is about 9.0 miles from central
Huntsville, AL.
The workforce of 30 produces Special trade contractors.
A Specialty Contracting leader, the company books about $4.6M.
Contact Mr. Burleson for more info about the company.

-------- {4400040} --------
Burleson Pool Company, Incorporated has an annual revenue of about $4.6M.
The company offers Special trade contractors in the Specialty Contracting industry.
Headquartered in Hampton Cove, AL, about 9.0 miles from central Huntsville, AL, the company
employs 30.
The company contact is Mr. Burleson.
```

Note that the individual sentences in each example contain different information, have different structures and are ordered differently. Yet each paragraph ultimately contains all the data, without repeating anything, and key information (such as the company name) is given before more secondary details (like the contact person).

(Also, of course, in a customer-facing product these sentences would most likely be formatted nicely into an HTML paragraph for easier reading—this sample output is broken up to allow easier comparison during testing).

For greater flexibility and future extensibility, the narrator is implemented using a "pluggable driver" pattern that can support extensive customization (see the Implementation Details below for more "documentation" of this code excerpt):

```
(defun construct-narrative (&rest resources &key pool table priority &allow-other-keys)
  (let ((pool (renderable-templates pool table))
        (selector (best-template-fn priority)))
    (apply #'narrate
           :pool pool
           :finished? #'narrative-finished?
           :required-keys (collect-narrative-keys pool)
           :chooser #'(lambda (&key pool &allow-other-keys)
                        (choose-any :pool (collect-best-items pool selector)))
           :pruner #'prune-redundant-templates
           resources)))
```

This “pattern” enables tuning the wide variety of potential output in various useful ways, for instance by prioritizing the placement of information (eg giving the company name in the first sentence), avoiding redundancy (eg saying “The company” rather than repeating the entire name) yet maximizing coverage by utilizing enough templates to present all available relevant information (eg mentioning the parent company if, and only if, a company is a subsidiary).

The flexibility the framework provides can be easily extended in the future (indeed, some of its current features were developed incrementally this way). For example to adapt to different use-case scenarios, the system might readily be provided with “brevity” or “style” parameters, favoring shorter/longer sentences or formal/casual language.

The generator can also be called either every time a user accesses the data—producing unique narratives each time—or the first output can be cached and thereafter retrieved, so that the text for a given company remains constant, but the whole corpus of “long tail” companies displays variety like the human-edited descriptions currently do.

Of course in addition to reporting company information, the narrative generator approach could obviously be deployed for use in other specific domains, such as people or products, simply by authoring appropriate templates and providing different input data.

Lastly, to demonstrate going beyond simply reporting existing information, an additional step was prototyped as a pre-process to the narration phase. In this step the system takes the input location information and uses it to *augment* the data set by adding the name of the nearest “major metropolitan center” (defined here to be cities of 100,000 population or more). Templates can then reference this derived datum in exactly the same way as raw data. This serves to illustrate how the fruits of more advanced forms of inferencing can be seamlessly integrated into the framework in the future.

Follow-On Development Options

There are a number of areas that this project has suggested for follow-on development of a more production scale system, if desired, including:

- Upgrade current “demo hardcoding” to make the system more flexibly configurable in general (eg data sources etc).

- Improve template authoring support, with features such as abstraction (“meta templates”), grouping or organizing related templates, and incorporating the templates themselves into a database.

- Evolve the “preprocessing” phase into a fully factored-out extensible framework module in its own right, including provision for both sophisticated inferencing but also simple data cleanup and normalization (some of the current demo data was hand edited in Excel; this should be better automated for production).

- Use (and as needed, help develop) better and more standardized ontology practices. For example the current set of template placeholder keywords is generated from the spreadsheet column headings—these should be replaced with OWL URIs and so forth.

- Explore adding more interesting information and inferencing, for example by exploiting federation with additional Hoovers and external data resources (eg product ontologies, social network data, etc).

Additional Implementation Details

The following rough description of the basic algorithm is taken directly from the comments in the code. It is not necessarily up-to-date or complete, but it conveys the essential ideas of the implementation.

```
;;;; narrative generation

;;; A backtracking system is used to generate narratives.
;;; The goal is, given a table (key-value map) and a pool (collection) of templates,
;;; to produce a narrative using some subset of those templates such that it is
;;;    1. Renderable: every key in the narrative has a non-nil value in the table
;;;    2. Concise: no key appears more than once in the narrative
;;;    3. Complete: every key in a designated set of required keys (eg "Big 7") appears

;;;  Initially, the narrative is empty and the pool contains some initial set of templates.
;;;  First, remove all templates which reference keys with null values
;;;  Then, repeatedly:
;;;    If the narrative is Complete, the process succeeds [but templates may be added if desired]
;;;    Next, effectively remove every template in the pool that duplicates a key in the narrative
;;;    If the pool becomes empty, the branch fails
;;;      else the pool contains templates: choose one (via some strategy passed as a parameter),
;;;      remove from the pool and add to the narrative

;;; The “Concise” constraint is to avoid redundant babble like repeating the company's name
;;; or restating the revenues multiple ways.  However it may be OK to substitute "pronouns" or
;;; "generic phrases", such as "the company" or "the revenues", depending on context and usage.

;;; [Note: it's not clear how reversible updates in the backtracker should best be implemented--
;;; Bindings which automatically track the computation, or an explicit "undo stack"?]

;;;; Narrate -- Basic search driver -- returns "proposal", a list of choices satisfying conditions
;;; Keyword API arguments used in this system:
;;;   "resources" -- refers to the entire keyword-rest list passed to some function
;;;   additional key arguments not mentioned here may appear to support passing them inward
;;; :proposal -- the proposal currently under consideration
;;; :pool -- a set of items available for adding to the proposal
;;; :finished? -- a predicate applied to the "resources" to see if the proposal is done
;;; :chooser -- sig (&key pool &allow-other-keys) function that chooses some element from the pool
;;; :pruner -- sig (&key pool &allow-other-keys) function that may return filtered reduced pool
```

The complete set of templates currently used in the demonstration prototype is given below. Of course in a real product there would probably be many more, and the template language and authoring system would doubtless be enhanced and extended to add features and support.

```
(:company_name " supplies " :line_of_business ", with an annual revenue of about $"
    :revenue_<$_million> "M.")

(:company_name " of " :primary_city ", " :primary_state " is a $" :revenue_<$_million> "M "
    :line_of_business " leader")

(:company_name " is a player in the " :primary_industry " segment")

("The company offers " :line_of_business " in the " :primary_industry " industry")

(:company_name " has an annual revenue of about $" :revenue_<$_million> "M")

(:company_name " supplies " :line_of_business)

("A leading vendor of " :line_of_business " is " :company_name)

(:company_name " boasts yearly gross sales of " :revenue_<$_million> " million dollars")

("A " :primary_industry " leader, the company books about $" :revenue_<$_million> "M")

("The company has an annual revenue of about $" :revenue_<$_million> "M")

("The company supplies " :line_of_business)

("Contact " :contact_prefix ". " :contact_last_name " for more info about the company")

("The company contact is " :contact_prefix ". " :contact_last_name)

("Headquartered in " :primary_city ", " :primary_state ", the company employs " :total_employees)

("This " :line_of_business " company has its home office in " :primary_city ", " :primary_state)

("The workforce of " :total_employees " produces " :line_of_business)

("The company's headcount is " :total_employees)

("Based in " :primary_city ", " :primary_state ", " :company_name " is " :total_employees "
    strong")

("Located near " :primary_city ", " :primary-state ", the company has " :total_employees "
    employees")

(:company_name " of " :primary_city ", " :primary_state ", " :near_metro " is a $"
    :revenue_<$_million> "M " :line_of_business " leader")

("Headquartered in " :primary_city ", " :primary_state ", " :near_metro", the company employs "
    :total_employees)

("This " :line_of_business " company has its home office in " :primary_city ", " :primary_state ",
    " :near_metro)

("Based in " :primary_city ", " :primary_state ", " :company_name " is " :near_metro)

("The company is " :near_metro)
```