TEMPLATE DESIGN FOR INFORMATION EXTRACTION

Boyan Onyshkevych
US Department of Defense
Ft. Meade, MD 20755
email: baonysh@afterlife.ncsc.mil

The design of the template for an information extraction application (or exercise) reflects the nature of the task and therefore crucially affects the success of the attempt to capture information from text. This paper addresses the template design requirement by discussing the general principles or *desiderata* of template design, object-oriented vs. flat template design, and template definition notation, all reflecting the results and lessons learned in the TIPSTER/ MUC-5 template definition effort which is explicitly discussed in a Case Study in the last section of this paper.

GENERAL CONSIDERATIONS

The design of the template needs to balance a number of (often conflicting) goals, as reflected by these desiderata, which apply primarily to object-oriented templates (see below), but also have applicability to flat-structure templates as well. Some of these desiderata reflect well-known, good data-base design practices, whereas others are particular to Information Extraction. Some of these desiderata are further illustrated in the Case Study section below.

- DESCRIPTIVE ADEQUACY - the requirement for a template to represent all of the information necessary for the task or application at hand. At times the inclusion of one type of information requires the inclusion of other, supporting, information (for example, measurements require specification of units, and temporally dynamic relations require temporal parametrization).
- CLARITY - the ability to represent information in the template unambiguously, and for that information to be manipulable by computer applications without further inference. Depending on the application, any ambiguity in the text may result in either representation of that ambiguity in the template, or representation of default (or inferred) values, or omission of that ambiguous information altogether.
- DETERMINACY - the requirement that there be only one way of representing a given item or complex of information within the template. Significant difficulties may arise in the information extraction application if the same interpretation of a text can legally produce differing structures.
- PERSPICUITY - the degree to which the design is conceptually clear to the human analysts who will input or edit information in the template or work with the results; this desideratum becomes slightly less important if more sophisticated human-machine interfaces are utilized, or if a human is not "in the loop". Using object types which reflect *conceptual objects* (or *Platonic ideals*) that are familiar to the analysts facilitates understanding of those objects, thus the template.
- MONOTONICITY -a requirement that the template design monotonically (or incrementally) reflects the data content. Given an instantiated template, the addition of an item of information should only result in the addition of new object instantiations or new fills in existing objects, but should not result in the removal or restructuring of existing objects or slot fills.
- APPLICATION CONSIDERATIONS - the particular task or application may impose structural or semantic constraints on the template design; for example, a requirement for use of a particular evaluation methodology or system for evaluation may impose practical limits on embeddedness and linking.

One other consideration comes into play when there is a current or potential requirement for multiple template designs in similar or disparate domains.

- REUSABILITY - elements (objects) of a template are potentially reusable in other domains; eventually a library of such objects can be built up, facilitating template building for new domains or requirements,

OBJECT-ORIENTED TEMPLATE DESIGN

The MUC3 and MUC4 terrorist domain templates were "flat" data structures with 24 slots; this led to considerable awkwardness in representing the relationships between data items in different slots. For example, in order to correlate the name of a terrorist target with the nationality of that target, a "cross-reference" notation had to be introduced. Additionally, large portions of the template would remain blank if there were no discussion of that type of information (e.g., if there were no human targets discussed at all).

In response to these difficulties, and in response to increased movement towards object-oriented data bases in Government and commercial applications, the template design for the TIPSTER/MUC5 task is object-oriented. In other words, instead of using one template to capture all the relevant information, there are multiple sub-template types (*object* types), each representing related information, as well as the relationships to other objects. A completed template is a set of filled-in objects of different types, representing the relevant information in a particular document. Each object thus captures information about one thing (e.g., a company, a person, or a product), one event, or an interrelationship between things, between events, or between things and events. A filled-in template for a particular document may, therefore, have zero, one, or many object instantiations of a given type. A completed template will typically have multiple objects of various types, interconnected by *pointers* from object to associated object. If there is no information in the document to fill in a given object, that object is not incorporated into the completed template. If a document is not relevant to the domain, no objects are instantiated beyond the "header" object which holds the document number, date of analysis, etc.

For example, both MUC5/TIPSTER domains had an object type ENTITY, which captured information about companies, organizations, or governments. Each company participating in a joint venture (in the JV domain) would be represented by a separate ENTITY object, with information about the NAME of the company (or government or organization), any ALIASES that are used to refer to it in the text, its TYPE (specifically COMPANY, GOVERNMENT, or ORGANIZATION), its LOCATION, its NATIONALITY (e.g., Honda USA Inc. is a Japanese company located in the US), pointers to objects representing PERSONs and FACILITYs associated with that company, as well as pointers to objects representing joint venture or parent-child relationships in which the company participates.

Although the task in MUC-5 and TIPSTER was to build a separate template for each document, the use of this object-oriented approach, and leveraging the current boom of object-oriented data bases and analysis tools, will facilitate the migration of this technology to a data base-building effort.

CASE STUDY: TIPSTER/MUC5

The template definition process in the TIPSTER/MUC-5 exercise consisted of a lengthy process of reconciliation of multiple, often contradictory, goals. In addition to the desiderata mentioned above (or an earlier, less well-understood version of that list), the templates needed to satisfy the programmatic goals of TIPSTER and the representativeness requirements of the participating government Agencies. The TIPSTER program was chartered to push the state of the art in Information Extraction in order to reach a breakthrough which would allow the wide-spread transfer of this technology to operational use; additionally, TIPSTER intended to chart out the capabilities of the technology.

To meet these goals, the tasks and templates were designed to (implicitly) cover a range of linguistic phenomena (e.g., coreference resolution, metonymy, implicature) and to (explicitly) require the full range of Information Extraction techniques (e.g., string fills, normalization, small-set classification, large-set classification). The task had to be structured in such a way that the management of the various funding Agencies would see that the technology had applicability to the type and size of tasks addressed by their Agency. This set of goals resulted in a need to define a set of tasks which would be substantially more challenging and extensive than the tasks from previous MUCs or current operational systems. Although still considered to be very substantial and extensive, the final template design reflect substantial trimming and reduction of information content from earlier versions, reflecting pragmatic programmatic considerations.

In the TIPSTER/MUC-5 exercise, templates were defined for two domains (see "Tasks, Domains, and Languages" in this volume). The template is defined in a BNF-like formalism which specifies the syntax of the template (the formalism is defined in Appendix A below); the semantics are defined in the Fill Rules document that was developed for each language/domain pair (see "Corpora and Data Preparation" in this volume).

The template that evolved over time didn't meet the Monotonicity desideratum in some cases. Although the "data bases" being built in the TIPSTER/MUC5 tasks were not dynamic over time, a small omission in a system template (vs. the "key" or answer template) at times reflected a Monotonicity failure in that the small omission resulted in major differences in the templates. For example, in the Joint Ventures domain, an ACTIVITY object could point to two (or more) INDUSTRY objects; however, if REVENUE (or START TIME or END TIME) information within that ACTIVITY were only applicable to one of the INDUSTRYs, that one ACTIVITY object would be split into two ACTIVITYs, each pointing to an individual INDUSTRY, along with any information specific to that ACTIVITY.

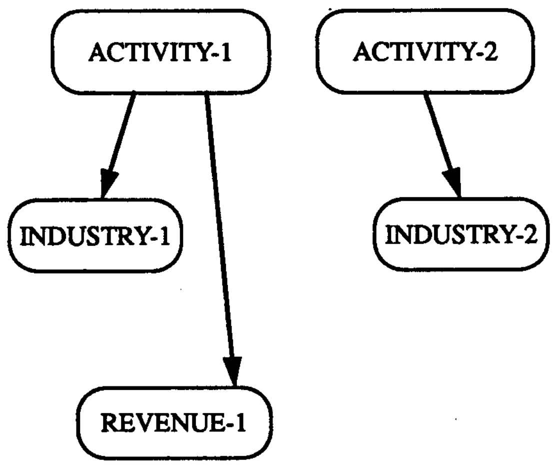

Figure1: Example of a correct template structure

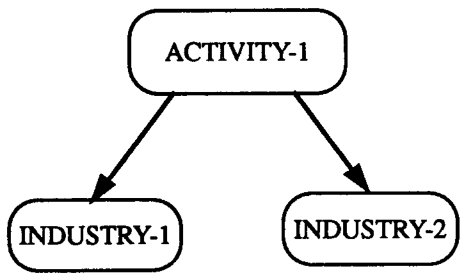

Figure2: Same template without REVENUE

Figure 1, for example illustrates how a (hypothetical) correct template structure piece might appear (diagrammatically); note two ACTIVITY objects. In Figure 2 (representing a template missing the REVENUE information) the omission of REVENUE information would not only result in a missing REVENUE object, it would also result in a spurious INDUSTRY fill on the ACTIVITY object (as well as an entire missing ACTIVITY object). Within the scope of the evaluation conducted in TIPSTER/MUC-5, this difference would result in a scoring penalty far greater than for one object.

In the TIPSTER/MUC-5 template for Joint Ventures, executives (and others) of the companies involved in the tie ups were represented in objects called PERSON, which represented the name and position of those individuals. Because the position information is not an intrinsic static property of that individual but rather transitory relational information (i.e., it reflects the nature of that individual's relation to a given company), the template design caused problems when the individual in question changed positions (often an executive of a parent company would become the president or director of a child company). Thus the Descriptive Adequacy desideratum was violated, since the template was not able to represent the change in that relationships between the individual and the companies. If we created a new object for a person for each position, we would violate the Perspicuity desideratum (since a PERSON object wouldn't represent a person *per se*, but a person in a particular job). Thus it would have preferable to either represent that relational information with the appropriate parameters (time *and* associated entity) or not at all.

A Determinacy desideratum inadequacy became apparent when it was noticed that the analysts who filled the templates had differing notions of how to represent multiple products in the JV domain. If two products, say "diesel trucks" and "four-door sedans" were to be manufactured as the ACTIVITY of a tie up, some analysts would instantiate one INDUSTRY object, then have multiple fills for the PRODUCT/SERVICE. Other analysts, however, would instantiate two INDUSTRY objects, put one product in each, then reference both INDUSTRYs from the same ACTIVITY. Although this was clarified in the Fill Rules, the analysts would occasionally err. A preferable solution would have been to allow only one PRODUCT/SERVICE per INDUSTRY, thus avoiding any possible Determinacy failure on this point (and ameliorating the Monotonicity failure discussed above).

APPENDIX A: NOTATION

< ... >	data object type (i.e., if indicated as a filler, any instantiation of that data object type is allowable). Every new instantiation is named by the type concatenated with: '-', the normalized document number, '-', and a one-up number for uniqueness. The angle-brackets are retained in the instantiation, as a type identifier/delimiter.
:-	what follows is the structure of the data object
:	what follows is a specification of the allowable fillers for this slot
::	what follows is the set itemization
{...}	choose one of the elements from the ... list. Note that one of the elements (typically "OTHER") may be a string fill where information which does not fit any of the other classes is represented (as a string); this set element would be identified by double quotes in the definition, and delimited by double quotes in the fill.
{{...}}	choose one element from the set named by ...(like {...} except that the list is too long to fit on the line)
#<... {...}#>	these delimiters identify a hierarchical set fill item. The first term after #< is the head of the subtree being defined in this term, and is itself a legal set fill term. What follows that term is a set of terms which are also allowable set fill choices, but are more specific than the head term. The most specific term specified by the text needs to be chosen. For example, the term #<RAM {DRAM, SRAM}#> means that RAM, DRAM, and SRAM are all legal fills; if the text specifies DRAM, then choose DRAM, but if the text specifies just RAM, then select RAM. In scoring, special consideration will be given when an ancestor of a term is selected instead of the required one (as opposed to scoring 0 as in the case of a flat set fill). Note that items in the set (i.e., inside the { ... }) can themselves be hierarchical item. Note that one of the elements (typically "OTHER") may be a string fill where information which does not fit any of the other classes is represented (as a string); this set element would be identified by double quotes in the definition, and delimited by double quotes in the fill.
+	one or more of the previous structure; newline character separates multiple structures
*	zero or more of the previous structure; newline character separates multiple structures; if zero, leave blank
-	zero or one of the previous structure, but if zero, use the symbol "-" instead of leaving position blank
^	exactly one of the previous structure
|	OR (refers to specification, not answers or instantiations)
(...)	delimiters, no meaning (don't appear in instantiations) NB: DOES NOT MEAN 'OPTIONAL'
((...))	delimiters, doesn't appear in instantiation, but contents are OPTIONAL but either all the contents appear, or none of them, in the case where there are no connectors (e.g., |) or operators (e.g., + or ^) within these delimiters: for example, with A ((B C)) D, only A D and A B C D are legal. If there is a connector inside these delimiters, then the either null or one of the forms are allowed fills: ((A | C)) means that the legal fills are 1) empty 2) A, and 3) C. Note that these delimiters essentially mean that the contents appear zero or one times. Also note that "OPTIONAL" here means that the position are left blank if no info, not that scoring treats these terms as optional.

..|..	Disjunction of the terms (XOR)
‘(‘	escape for the paren (i.e., the paren appears in the slot fill in that position)
‘)’	escape for the right paren
“ “	any string (from the text, except for COMMENT fields). The quotes remain in the instantiation around non-null-string fills.
“...”	any string (from the text); the ... may be a descriptor of the fill. The quotes remain the instantiation around non-null-string fills.
[...]	normalized form (see discussion for form specifications).
[[..]]	range; select integer from specified range; left-pad integer fills with 0’s, if necessary, to conform to number of digits used
/	This notation is for answer key templates only (test or development), not for system answers. The slash indicates a disjunction (XOR) of allowed answers. Each disjunct appears on a new line. If the / appears as the first character of a slot filler, then a null answer (i.e., no fill) is an allowable fill. If multiple fillers are allowed (by a + or * notation) for the slot, then the possible fillers are given in disjunctive normal form (variable number of conjuncts per disjunctive term), for example, (disregarding the new-lines): / NICHROME GOLD / NICHROME GOLD TUNGSTEN TITANIUM would mean that the three allowed answers are 1) (empty string),2) NICHROME GOLD, and 3) NICHROME GOLD TUNGSTEN TITANIUM. An object can be indicated as being optional if (all) pointers to that object appear after a /. System answers are not allowed to offer optional or alternate fills (answers).

Unless otherwise marked (i.e., by +, -, or ^), a slot may be left blank if the information is absent in the text. If a structure descriptor is not terminated by +, *, -, or ^, then zero or one of the structure are allowed. If two (or more) structure descriptors are given without a connector between them and without either one being marked by +, *, -, or ^, then either both appear or neither appears: [NUMBER] ‘C’ means that 423 C is a legal fill, but 423 is not, nor is just C.