ISSUES AND METHODOLOGY FOR TEMPLATE DESIGN FOR INFORMATION EXTRACTION

Boyan Onyshkevych

Department of Defense
Attn: R525
Fort George G. Meade, MD 20755
baonysh@afterlife.ncsc.mil

ABSTRACT

The goal of Information Extraction tasks is to identify, categorize, classify, relate, and normalize specific information of interest found in free text, and to make that information available to a back-end data base, data fusion, or other application. A data structure referred to as a *template* is typically used for capturing such information, particularly in cases where the amount and complexity of information is substantial. The design of the template for such an application (or exercise) thus defines the task itself and therefore crucially affects the success of the Information Extraction attempt.

This paper discusses template structure and methodological issues which arise in the template design process, within the context of a discussion of the design process itself; this paper is based on the template design process for TIPSTER/MUC5 and certain subsequent Information Extraction exercises. The first section of this paper addresses the issue of selection of the appropriate data representation (text annotation vs. flat template representation vs. object-oriented template). The second section outlines a set of high-level design considerations (*desiderata*) that have emerged; these desiderata feed into the discussion of design elements and a procedural review of the design process (design iterations, use of those linguistic analysis tools, etc.)

1. Data Structure Selection

Although the selection of an appropriate data structure for representing extracted data may be influenced by the data structure requirements of the back-end application, the use of straightforward deterministic data format converters can further decouple those two data structure requirements. Thus a data structure can be selected to be appropriate for the data extraction task itself.

The data structures for Information Extraction fall into three broad categories: text annotation, flat data templates, and object-oriented templates. The appropriateness of those three formats to a particular task is primarily based on the richness of the required data complex.

If a task calls for a small number of primitive data types, with no requirements for representing interrelations among primitive data types, *text annotation* may be the simplest representation. This data structure is renderable as tagging delimited text segments with appropriate tags from SGML or another mark-up language (or, equivalently, by an auxiliary file for each document with the tag associated with an offset into that document file). For example, the data from the task of finding company and product names in a text may be most appropriately represented by an annotation scheme. However, if the task also requires the identification of coreferences among names or references in a text and/or association of other attributes of those elements, a template structure may be more appropriate.

Flat templates, such as those used in MUC3/MUC4, associate related data elements (either strings from the text, categorization of data, or normalized data). Each such template thus represents a data complex of related information; each complex of data from the text will result in another template (with the same structure) being instantiated. A flat template's structure is thus a set of slots (naming the attribute), each with zero, one, or more possible fills (such as strings from the text, numbers, or symbols from a predefined set).

The MUC3/MUC4 templates were flat data structures with 24 slots; there was a requirement to represent relationships between data elements in different slots, which led to some awkwardness. 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.

In response to such difficulties and because of the richness of the required data complex, the data structure for tasks such as the TIPSTER/MUC5 task is most appropriately *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 (or *instantiated*) template is a set of filled-in objects of different types, representing the relevant information from a particular document. Each object thus captures information about one thing (entity), an event, or an interrelation between other objects, A filled-in template for a particular document may, therefore, have zero, one, or more 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 given document is not relevant to the domain, no objects are instantiated (possibly beyond a "header" object which holds the document number, date of analysis, etc.

2. Design Desiderata

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 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.

- 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. Perspicuity is facilitated by enforcing separation of event, entity, and relational information; for example, instead of having a `buyer` object and a `seller` object in a sales event (where both are companies), having a `company` object more closely parallels the conceptual kind (the roles of the companies would be reflected by the semantics of the slots that point to them in the `sales` event object).

- 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. Violation of this desideratum may lead to "keystone" effects, where one missing item of information results in a radically different template structure.

- SINGULARITY - this requirement states that a real-world entity or event maps to only one element in the template, and that if it plays multiple roles, pointers are used to that one element. When viewing an instantiated template as a graph (objects and fillers as nodes, slots as arcs), singularity states that there should be only one node representing a specific real-world entity, relation, or event. Note that in some cases, where time is a critical parameter and the template tracks a dynamic situation, time may be associated with a particular entity, event, or relation; objects with different time indicators effectively identify different referents and thus map to different objects (the ACTIVITY object in the TIPSTER Joint Venture template illustrates this). Unlike this recommendation against one-to-many mapping, the converse situation may occur, but needs to be carefully monitored to minimize monotonicity violations. For example, if a group of 39 companies together plays a certain role, it may be impractical and unnecessary to independently represent each one; but one may need to be singled out and hence be separately represented.

- 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.

3. Design Elements

In addition to any ancillary supporting materials required by the domain (such as gazetteers or name lists), three definitional documents or knowledge sources provide the information necessary to define the syntax and semantic of the template and to define the process of filling it.

3.1. Template Definition

The basic syntactic definition of the template defines the structure of the template, including specification of all object slots and type definition of legal slot fillers. For those slots which are filled by pointers, an indication of the legal types of the pointer referents is included; similarly, for slots which contain set fills (or classifications or categorizations from a finite set of categories), the set of possible fills is defined by enumeration. For TIPSTER and some other subsequent Information Extraction tasks, a BNF-like template definition language is utilized (see Appendix below).

As part of a support effort to TIPSTER, the Computing Research Laboratory of New Mexico State University produced a graphical interface-driven tool to support the definition of a template; the tool produces not only a BNF definition for the new template design, but also compilable source code for a MOTIF-based tool for manually filling in templates (along with supporting routines).

3.2. Rules of Interpretation

The semantics of the template are defined in another document (this was called the Template Fill Rules document in TIPSTER). In addition to providing definitions of various terms or concepts, the Rules of Interpretation (ROI) document presents *reporting conditions*. The document needs to specify when anything at all is to be instantiated for a given document. Then reporting (i.e., instantiation) conditions need to be specified for each object type. The conditions specify when there is enough information in the text to instantiate the particular object; this may be defined in terms of how information appears in the text (e.g., centrally vs. peripherally), or a specification of a minimum number of slots that need to be filled in order for the object to be valid. The semantics of each object type is also specified in the ROI.

For each given slot, then, the reporting conditions are specified. The ROI defines the semantics of each slot, as well as detailing the specification of legal fills (and the translation from text to fill format). For set fills, the ROI defines each symbol from the set and specified reporting conditions. For string fills, the extent of the string (i.e., what elements from the text get included in the string?) is defined along with any normalization that is to be done on the string.

3.3. Case Law

Since the definitions in the ROI are not likely to capture every possible eventuality (even the most diligent case), they are supplemented by a set of examples. These examples may either be incorporated into the ROI document in each appropriate section, or compiled into a separate document or collection. The template designer should not expect that this collection becomes fixed, because as new language usage, new types of information, or previously unseen reportable event types occur, the case law collection needs to be increased to document the analyst's handling of the new data and to ensure conformity for future occurrences. The examples that need to be added to this collection include any situations where it is not perfectly clear how the rules in the ROI apply, as well as cases which are fairly complex and where having a guide helps the analyst in constructing the template. Violations of the desiderata above (particularly determinacy) increase the need for a case law collection.

4. Design Methodology

The design process of an appropriate template structure for a complex task is necessarily an iterative one. After an initial sketch, the template elements should undergo tuning based on corpus analysis. Then the template is subjected to iterative refinement based on difficulties and novel inputs encountered while filling a number of templates manually.

4.1. Template Sketch

An initial template sketch is devised to reflect the task requirements as understood by the customer, and constructed adhering to the desiderata above. If available and appropriate, objects from a template object library (or from previous template designs) are utilized in this initial template draft, with unnecessary slots being pruned.

In the object-oriented data structure paradigm, objects typically fall into one of three types: entities, events, and relations.

- *Entity* objects represent a conceptual object and its attributes; the objects typically represent some type of real-world entity such as a person, an organization, a product, a company, etc. Entity objects may also be used to represent such things as times and locations (in isolation) that other entities, events, or relations may point to; for example, a location object may include various types of information about a place (coordinates, name, elevation, etc.) and may be pointed to by an organization as its location, by a transportation event as the destination, etc.

- *Event* objects represent real-world actions or processes. The event object will typically have pointers to objects representing the participants in the event, and may include slots representing the parameters or results of the event. A typical example may be a `sales` event object with pointers to the `buyer` and `seller` (each represented by an object).

- *Relation* objects reflect relations between entities, or relations between events; relations between an event and an entity are typically reflected by a pointer in a slot on an event object. Relations are often collapsible into a slot/pointer representation, but there are some compelling reasons to retain them as separate objects instead. An example of an entity/entity relation is the relation between a `person` object and a `company` object which specifies the role (such as `President` or `CEO`) that the person has in that company. In a task setting where tracking the changes in leadership of companies is important, instead of maintaining an `officers` slot on a `company`, (or a `position` slot on a `person` object) a relation object is preferred, with pointers to `company` and `person`, an indicator of the position, and a time stamp (to capture change).

In the template, a given event, entity, or relation from the domain may either be represented by an appropriate object, as described above, or may be collapsed into a slot value (attribute) of another object. Typically if only one or two elements of information about a particular entity, event, or relation are needed, it is more expedient and concise to collapse that potential object into a single or multiple-valued slot. For example, if an object representing a company captures the headquarters location by the place-name only, (and no further geographic or gazetteer information is necessary), then a slot on the company object with a simple string fill is preferable to a pointer to a separate location object which just represents the name of the location.

Two or even three elements of associated information can be treated as a composite slot fill instead of a two- or three-slot object. One strong reason to maintain a cluster of associated information as a separate object (even if it only consists of one, two, or three elements) is the singularity desideratum. For example, a person may have multiple roles in a particular template, and the only information that is maintained about that individual is the name. In this case it may be worthwhile to maintain a distinct `person` object with that information, even if it only has one slot. This mechanism explicitly indicates coreference of the multiple person

references in the various roles, instead of relying on string equality to indicate coreference. Perspicuity may be increased by this approach, despite the proliferation of objects, because of the intuitive correlation of one template object with one real-world object.

4.2. Corpus-Based Tuning

This initial template is augmented or further pruned to reflect the data. If a certain item of information is not found in a substantial sample of the text corpus, and it is not a critical datum, pruning is indicated. Data categorization or association of multiple data elements is often more complicated than initially envisioned; for example, the list (with percentages) of owners of a company is not static, therefore either a reference time is specified by fiat, or that information is associated with a time data element.

Tools such as KWIC, mutual information, or n-gram analysis can be utilized to identify the typical context of relevant information elements in a text corpus. For example, KWIC on 'joint venture' helps identify the different activities or situations that joint ventures appear in, thus identifying possible renderable data elements relating information about joint ventures in a template.

The steps below identify one possible path in tuning a draft template to the corpus.

- Identify the key entity types in your requirement; find out typical ways that those data elements are expressed in the text. For example, if companies are key entities, the corpus may reveal that companies may be referred to by name (in which case markers such as Inc. and Ltd. identify some occurrences) or by definite reference ("the company" or "the manufacturer" may identify such). It is not necessary to identify all the different ways the entity can be referenced at this stage; however, in some cases it will be possible to easily enumerate a large percentage of the ways in which something is referenced.

- Using these tags or markers, find the references in a non-trivial set of documents (100 or more) to that entity. Evaluate the context in which these references occur; which of the semantic contexts are of relevance to the domain? Any contexts that have not been addressed but are of interest or are necessary for coherence of the representation need to be added to the template definition. This analysis will help evolve the event and relation object definitions. For example, in the Joint Venture template, such analysis of the corpus revealed examples where the joint venture "expanded" or "increased", and the decision was made to add that situation to the `status` set fill list.

- Now given the contexts identified above (and marked, for example, by specific verbs), search the corpus for the occurrences of those markers and identify the contexts. An examination of those text fragments will reveal two things: 1) other ways that the entities may be referenced in the text (such as indefinite or generic references) and 2) other entity types that can participate in the same contexts as the entities of interest. An evaluation of the former will help determine the reporting conditions for the entity, while an evaluation of the latter is necessary to determine which of the new entity types should be reported. For example, in the Joint Venture corpus, this analysis reveals that consortia appear in the same roles as companies and governments, and a determination was needed as to their reportability and the categorization of that type of entity.

- This process may be iterated, and repeated on each entity/event or entity/relation of interest in the template.

- Techniques for identifying information particular to the given domain include finding differentials between the word or n-gram frequency lists for the domain corpus and for a general corpus. Any term that appears more in the domain corpus (vs. a general corpus) needs to be evaluated for inclusion in the template; in some cases the terms identify relevant concepts, in others these concepts are beyond the scope of what needs to be tracked. However, some of the concepts or terms that are of relevance (such as temporal expressions or common actions) will be equally frequent in various corpus types, in which case this technique will not identify them. This technique is similar to New Mexico State University's statistical filter for TIPSTER.

- In general, for each newly-identified data type (to include classes in a set-fill), a decision needs to be made: 1) don't report it at all; 2) report it by coercing it into an existing data type, (e.g., declaring that consortia are the same class as companies); or 3) expand definition to handle new data type (e.g., adding a consortia class to the set fill list).

4.3. Iterative Refinement

The template undergoes a cycle of further refinement through manual filling of the template based on a substantial number of documents; based on the complexity of the template and diversity of text types and sources, the number required for this cycle could be 300 or more. In fact, the template could be subject to modification throughout the lifetime of the task (based on novel inputs), but typically operational stability will require freezing the template definition; the ROI may be subject to update to reflect the coding decisions made on novel inputs, particularly if that input may be expected to reoccur. In an operational environment, such ROI augmentation may still be conducted to reflect new inputs, so long as care is taken to avoid any changes affecting previously filled documents. In fact, in order to maintain consistency (if there are multiple systems and/or human analysts creating or modifying template instantiations) such changes to the ROI and especially the case law collection are desirable.

When a change is made to the template or ROI, the existing body of filled templates may be affected. In some cases the appearance of a new type of relevant information may require reworking the existing template structure or the partitioning reflected in a set fill list; in such case, the impact on the template corpus is substantial, and a decision needs to be made whether to update the older versions (depending on operational need). In other cases, an addition doesn't impact the corpus at all, for example, when adding a previously-unseen currency type to the set fill list of currency types. In our experience, different analysts interpret rules in the ROI with different degrees of strictness, which may lead to new information types escaping unnoticed; for example, some analysts might treat

consortia as companies without giving it a second thought, whereas others would identify that they are not companies, strictly speaking, and require a ROI or case law clarification of that situation.

At occasional points in the iteration, the template should be reviewed in its totality for violations of the desiderata. When violations are identified, manually running through some (reasonable) worst-case scenarios will help identify whether those violations will cause problems, thus should be addressed, or whether restructuring would cause more problems (e.g., in perspicuity) than leaving the structure as is.

The corpus-based tuning and iteration process described above directly addresses the descriptive adequacy desideratum. The determinacy desideratum can also be addressed by the iteration process, in particular by using more than one analyst to independently create a set of templates for the same set of documents; some of the discrepancies between the independent codings will highlight determinacy violations. Additionally, the independent codings may identify perspicuity violations (where an analyst did not understand the template structure or notation).

The effects of violations of some of these desiderata in the TIPSTER templates are discussed in "Template Design for Information Extraction" in the proceedings of the TIPSTER program, as well as in the Proceedings of the Fifth Message Understanding Conference.

5. Appendix: BNF for Template Definition

Except as specified, the notation below is for the template definition.

`< ... >` 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 for template definitions, or the contents of the instantiated object for instantiated templates/

`:` what follows is a specification of the allowable fillers for this slot in a template definition, or the filler of the slot in an instantiated template.

`::` what follows is the set itemization in the template definition.

`{...}` 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.

6. REFERENCES

1. *Proceedings of the TIPSTER Text Program, Phase One.* forthcoming.

2. *Proceedings of the Third Message Understanding Conference (MUC-3).* San Francisco: Morgan Kaufman, 1991.

3. *Proceedings of the Fourth Message Understanding Conference (MUC-4).* San Francisco: Morgan Kaufman, 1992.

4. *Proceedings of the Fifth Message Understanding Conference (MUC-5).* forthcoming.