PRINCIPLES OF TEMPLATE DESIGN

Jerry Hobbs, David Israel

Artificial Intelligence Center
SRI International
Menlo Park CA 94025

ABSTRACT

The functionality of systems that extract information from texts can be specified quite simply: the input is a stream of texts and the output is some representation of the information to be extracted. Hence, the problem of template design is an instance of the problem of knowledge representation. In particular, it is the problem of representing essential facts about situations in a way that can mediate between texts that describe those situations and a variety of applications that involve reasoning about them.

The research on which we report here is directed at elucidating principles of template design and at compiling these, with examples, in a manual for template designers.

1. Introduction

The functionality of systems that extract information from texts can be specified quite simply: the input is a stream of texts and the output is some representation of the information to be extracted. In the message understanding research promoted by ARPA through its Human Language Technology initiative, the form of this output has been templates (feature-structures), with complex path-names (slots) and various constraints on fillers. The design of these templates, especially considered as concrete data structures, has been determined to some degree at least by considerations having to do with automatic scoring. Beyond that, it has not been made clear what principles have driven or should drive the design of these output forms; but it has become clear that serious defects in the form of the output can undermine the utility of an information extraction system. If the output is unusable, or not easily usable, the breadth and reliability of coverage of the natural language analysis component will be of little value.

As part of the **DASH** research project on Data Access for Situation Handling, we are attempting to elucidate principles of template design and at compiling these, with examples, in a manual for template designers. Our methodology has included detailed critical analysis of the templates from a variety of information extraction tasks (MUC-4, MUC-5, Tipster-1, the Warbreaker Message Handling [WBMH] tasks), together with the creation of templates for the TREC topic descriptions and narratives.

The design of templates, or more generally, abstract data structures, as output forms for automatic information extraction systems must be sensitive to three different but interacting considerations:

1. the template as representational device
2. the template as generated from input
3. the template as input to further processing, by humans or programs or both.

The central consideration in our research is that of the template as a representational device. The problem of template design is a special case of the general problem of knowledge representation. In particular, it is the problem of representing, within a constrained formalism, essential facts about situations in a way that can mediate between texts that describe those situations and a variety of applications that involve reasoning about them.

What facts about a situation are essential is determined by a semantic model of the domain, which is in turn motivated by the particular information requirements of the analytical purposes which the extracted information is to serve. This specification could, in principle, be done without any detailed thought given to the nature of the texts from which information is to be extracted; thus it could include information requirements that simply could not be met by the input stream. It might also abstract from information readily transduced from the input stream. Conversely, the domain specification may reveal cases where one must extract information that is not important to the end user in order to disambiguate or otherwise explicate important informational content. Again, the domain model could be specified without any detailed thought given to the design of the concrete syntax of the template. In this latter regard, crucial considerations include intelligibility and 'browsability', together with the utility of the template fills as input to further processing.

We here report some results of a program of research aimed at uncovering the underlying principles of template design.

2. Basic Ontology

In constructing a representation for a domain or task, the first questions to ask are:

1. What are the basic entities? What properties of these objects and what relations among them are we interested in?
2. What kinds of changes in such properties and relations are we interested in?

Answers to any one of these questions depend on answers to the others. Answers to the first provide the basic *ontology* of

the representation.

Basic Entities The basic entities should be things that endure throughout the temporal focus of the task.[1] They enter into the relations and are characterized by the properties of primary interest and are the participants in events that may change those properties and relations. In the joint ventures domain, companies are the primary candidates for basic entities. In the long run, they get formed, split, merge, and go out of business, but for many analytical purposes, and in particular for the purposes implicit in the MUC-5 task, we can think of them as permanent. It is companies that enter into joint venture relationships and through such relationships bring about the one crucial exception to the rough-and-ready rule just mentioned: the creation of new, joint venture companies. In the same domain, facilities and people are also good candidates for basic entities.

The basic entities may be represented by structured objects with a number of slots, as follows:

```
<TEMPLATE> :=

        ........

        COMPANY:          <COMPANY-1>

        ........
<COMPANY-1>:
        Name:       ''General Motors''
        Nationality:       U.S.
        ........
```

or by an atomic element such as an identifier, a set fill, a number, or a string:

```
<TEMPLATE> :=
        .......

        COMPANY:          ''General Motors''

        ........
```

The difference in outcome between these two cases is that in the former you have to look elsewhere for the information about the entity, whereas in the latter you don't. In general, it's better not to have to, so unless there is a good deal of information that needs to be recorded about the type of entity in question, it is better to use an atomic element to represent such entities. Again, within the joint venture domain, companies are good candidates for representation as structured objects, since we need to know their aliases, location, nationality, officers, etc. On the other hand, within that same domain, it may be that the only information we need to record about a person, aside from his relation to a company, is his name, so in that case it is better to represent the person (atomically) by his name.

[1] For more on this, see next section.

Natural Kinds It is better if the types of basic entities, especially those represented by structured objects, are 'natural kinds', that is, if they correspond to fairly natural, intuitive ways of classifying and characterizing the domain of interest. For example, companies, people, facilities are natural kinds in this sense. Ordered pairs of Industry Types and Product/Services are not. Rather than have basic entities of unnatural kinds, one may opt for more, or more complex, slot fills in objects of more natural varieties. Still, it should be remarked that one's commonsense demarcation of a domain into basic entities is always subject to revision by the particular analytical demands of the task at hand. Thus, in the case of WBMH, while units (e.g., divisions, battalions, etc.) are a perfectly natural kind of entity, deployments, that is relatively short-lived activities involving elements from units, may be less natural but they are at least equally central.

Associating Properties with the Right Objects It is important to determine whether the property encoded in the slot of an object is really a property of that object, rather than of some other related object. For example, in the Tipster templates, Total Capitalization was viewed as a property of the Contribution object, whereas it is really a property of the Tie-Up Relationship, and thus should be associated with that object. This misplacement of properties seems especially likely when the entities in question are types of relationships or activities, as they are in this case. We return to the issue of representing relations below.

3. Temporal Granularity

We have noted that the issue of what kinds of changes are of interest relative to a given task is centrally important to the design of templates for the task. The resolution of this issue is a crucial determinant, in particular, of what we call the *temporal granularity* of the representation. Certain properties of and relations among entities are relatively permanent; others are relatively short-lived. But what counts as permanent and what as short-lived is itself dependent on our interests and purposes, both theoretical and practical. An analysis of the kinds of changes that are of interest should determine, even if only roughly, a temporal interval or length of time as its focus or window. See Fig. 1. Note that there is a mutual dependence here: Properties and relations that are apt to change within that time interval are temporary; those that are likely to hold throughout the designated interval are, with respect to this task, permanent. Thus, the fixing of a temporal granularity allows the resolution of many problems in template design by defining limits on what we have to specify.

For example, in the joint ventures domain, we are interested in the formation (or dissolution) of tie-up relations among companies. Thus such relations are temporary, whereas subsidiary relations are permanent. If we were interested in buy-outs, subsidiary relations would be viewed as temporary, changes in such relationships being an important focus for the task. In the domain of troop movements or deployments, locations and associated equipment are temporary, whereas a unit's place in the command hierarchy is permanent, even though on the scale of decades (or even much less), that might change.

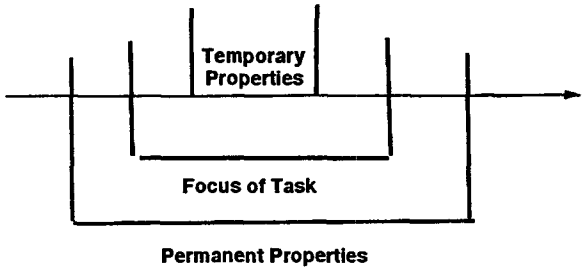

Figure 1: Temporal Granularity and Focus

Note that temporal granularity is task-relative rather than message-relative. The messages may have been written from very different temporal perspectives, with very different interests and purposes. We need to extract the information from them in a form that is appropriate for the task at hand.

4. Representing Relations

A relation can be represented in one of two ways, as a separate object in its own right, or as a property of one of its arguments. See Fig. 2

For example, the subsidiary relation could be represented by its own Entity Relationship object, or it could be represented by a Parent Company slot in the Entity object.

The following criteria seem useful in deciding which of these options to adopt:

1. If the relation is of primary interest in the task, option (a) may be the best choice.
2. If a lot of other information needs to be recorded about that relation, option (a) is a good choice; if only the two arguments need to be recorded, option (b) is probably better.
3. If the relation is permanent relative to the temporal granularity of the information task, then option (b) is a good choice.
4. If some other relation, $Relation_2$, depends on $Relation_1$, in the sense that the former cannot exist without the latter existing, then $Relation_2$ is a good candidate for being represented via option (b).

With respect to the second criterion, if in addition to the two arguments, we want to specify the time, the location,

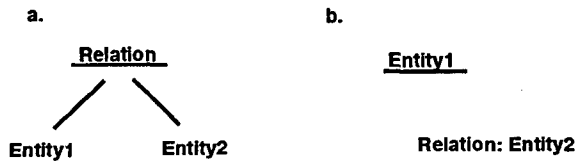

Figure 2: Representing Relations

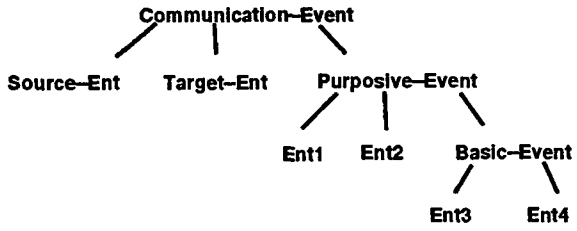

Figure 3: Typical Event Structure

and various other aspects of the relation, then option (a) is indicated. With respect to the third criterion, if the relation is at least as permanent as the entities, then option (b) is a good choice. These two criteria overlap to some extent. If the relation is permanent, there is likely no need to record its time.

In the specific case of the Subsidiary relation in Tipster, it is not the relation of primary interest (Tie-Ups are), there are no other properties that need to be specified for the relation other than the parent and child companies, and the relation is permanent with respect to the temporal focus of the task. Therefore, option (b) seems appropriate.

The Tipster template presents an apposite example of criterion 4 as well. A Contribution, as conceptualized in the template, is a relationship, just as a Tie-Up-Relationship is, so it certainly could qualify for object status. However, it is dependent on a Tie-Up-Relationship; a Contribution relationship among companies can't exist without a Tie-Up-Relationship among them. This indicates option (b) is appropriate.

5. Events

We can classify events, and the relations among entities that they involve, in different ways for different purposes. On the basis of an examination of a variety of templates, we hypothesize that there are three central event types. First, there are those that directly relate two or more basic entities, such as a company manufacturing a product or a terrorist organization attacking a target or a vendor supplying a buyer with a part. These very same events, however—especially if, as in the examples just mentioned, they involve purposive agents—can also be classified in terms of their purpose or aim. This type of classification typically involves further reference to an activity or condition, as when a company manufactures a product *in order to* enter a new market or when two companies form a joint venture *for the purpose of* carrying out some activity. Third, there is the specially important type of event involving communicative relations among basic entities, together with a content communicated, itself comprising some further activity or event of any of the three types. Thus, a typical event structure might be represented as in Fig. 3.

Of course, in many cases there would be equations identifying the various entities involved. Thus, GM might announce it is forming a joint venture with Toyota for the manufacture of cars by GM in Japan, where Source-Ent = Ent1 = Ent3 = GM. We also note that a Communication-Event can have a Basic-Event for its third argument.

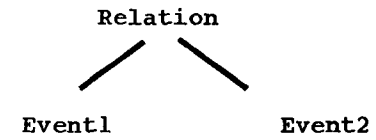

Figure 4: Relations Between Events

In addition to these three event types, there are relations between events that we may need to represent, such as causality or the subevent (part-whole) relation, as in Fig. 4. Thus, a shooting event could cause a dying event, and a troop movement might be part of a larger attack.

In general, the template structure should be no deeper than this. It is better for the trees to be very broad (i.e., for individual objects to have lots of slots) than to be very deep.

6. Entity Snapshots

In many applications, there are a large number of temporary or transient properties of entities that are of primary concern. If we design the template around the enduring basic entities themselves, it might seem that these temporary properties should be demoted to mere slots rather than be represented as entities in their own right. These slots, on the other hand, would also have to allow multiple entries and each entry would have to have time stamps. A way to eliminate this complexity is to have as first-class objects, in addition to Entities, Entity Snapshots. An Entity Snapshot is an Entity at a particular point or interval in time. As such, an Entity Snapshot would have a pointer to the Entity that it is a snapshot of. It would also carry all the temporary information about the Entity. The time of the snapshot would also be one of the slots.

In the WBMH domain, these Entity Snapshots, under the name Entity Information, are primary objects of interest. They represent deployments, or "target opportunities". Such temporary properties of Entities as Equipment, Location, Direction, and so on, are really to be associated with deployments, Snapshots, rather than Entities or Units.

6.1. Entities from Entity Snapshots

Often the first way one might think of an entity is in terms of its structure and properties at a particular moment in time. One later realizes that in fact the entity maintains its identity over time as its internal structure changes. In this case we should reconceptualize the entity as being a mapping from instants or temporal intervals into its structure and properties at that time.

For example, one's first intuition about the nature of a department may be that it is a set of employees. Later one realizes it should have been conceptualized as a mapping from times to sets of employees. In this case, it is a good idea to have both Departments and Department Snapshots, where the set of employees is a property of the Department Snapshot.

There are a number of interesting problems of analysis that revolve around the relationship between entities and entity snapshots. Sometimes one is of primary interest, sometimes the other. For example, in Desert Shield, units were of interest; in particular a major focus of concern was the calculation of unit strengths. In Desert Storm, however, deployments were of primary interest, since it was deployments that presented the immediate danger.[2] In general, we want to be able to infer the identity of different deployments across time, to infer their membership in units, to derive some of their properties from default properties of their units, and to determine properties of units, such as unit strength and readiness, from properties of deployments.

7. Slot Fills

Slot fills should be uncomplicated. They should take one of the following forms:

(a) Atomic elements, such as identifiers, numbers, or strings.

(b) Pointers to structured objects.

(c) Tuples whose elements are of types a and b.

(d) Sets whose elements are of types a, b, or c.

It is probably confusing to have tuples with more than three elements. Thus, the maximum complexity of a slot fill would be

$$\{\langle A1, B1, C1\rangle, \langle A2, B2, C2\rangle, \ldots\}$$

Many set fills of type (d) whose elements are of type (c) may be thought of as functions. For example, if we had a set of pairs of companies and ownership percentages, we could think of it as representing a function from companies to ownership percentages. However, not all set fills of this type are conveniently thought of as functions. If we have an Officers slot for the Company object, whose filler is a set of tuples of the form ⟨Position Person⟩, then an entry might be:

{⟨PRES, "White"⟩,
⟨CEO, "White"⟩,
⟨SREXEC, "Brown"⟩,
⟨SREXEC, "Green"⟩}

This is a not function in either position or name.

7.1. Objects or Tuples

It is of course possible, and often good programming practice, to implement tuples as structured objects:

```
Tuple:
        A:
        B:
        C:
```

[2] This pair of examples also illustrates that different but intimately related tasks can have different temporal granularities.

But in the presentation of the templates, it is often better from the user's point of view to represent them as tuples, rather than multiplying kinds of objects. This is an instance of the principle that the user shouldn't have to go looking too far afield for information. As you follow a complex path of pointers, it can be easy to forget what the type of an object is and where it fits into the web of relationships you're interested in.

8. Backpointers

Memory is cheap. Time, especially the user's time, is expensive. Therefore, a user should be able to browse through a database, easily traveling from any node to any related node. In a troop movement domain, sometimes the user will want to ask the two questions,

1. What activities are going on in Sector A?
2. What units are involved in these activities?

and sometimes he will want to ask:

1. What activities is Unit A involved in?
2. What is the location of these activities?

Therefore, for every pointer from one object to another, there should be a backpointer.

It might be objected that backpointers amount to storing information redundantly, but that's rather like saying two-way streets are redundant because you can always get back to where you started by some other route. However, backpointers should be considered secondary. They do not need to be part of the template definition. It should just be assumed that the backpointers will be constructed as well. (Moreover, in evaluations, backpointers should not be scored. This was one of the chief difficulties in the scoring of the Tipster templates.)

9. Summary

We have reported on research directed at elucidating generally applicable principles of template design. The guiding perspective of the research reported here is that template design is a special case of knowledge representation in a constrained representation language. Thus it is no surprise that many of the main issues in knowledge representation, issues of choice of ontology, of the nature of relations and of events, arise here as well. We have also paid attention to issues of readability as well, for if the templates produced, either by hand or by program, are not easily intelligible, their accuracy and completeness will be of little use.

10. Acknowledgements

The **DASH** project has been sponsored by the Office of Research and development, under Contract No. 93-F149300-000. We would especially like to thank William Schultheis of ORD for his active and extremely useful participation in the research. We would also like to thank Boyan Onyshkevych, especially for discussions about the Tipster template, and Mabry Tyson.

References

1. Sundheim, B., ed. *Proceedings*, Fourth Message Understanding Conference (MUC-4), McLean, VA, June, 1992. Distributed by Morgan Kaufmann Publishers, Inc., San Mateo, CA.
2. Sundheim, B., ed. *Proceedings*, Fifth Message Understanding Conference (MUC-5), Baltimore, MD, August, 1993.