Next: Exam-like questions **Up:** Structured Objects **Previous:** Semantic Nets

Frames

Frames are a variant of nets that are one of the most popular ways of representing non-procedural knowledge in an expert system. In a frame, all the information relevant to a particular concept is stored in a single complex entity, called a frame. Superficially, frames look pretty much like record data structures. However frames, at the very least, support inheritance. They are often used to capture knowledge about *typical* objects or events, such as a typical bird, or a typical restaurant meal.

We could represent some knowledge about elephants in frames as follows:

```
Mammal
  subclass:      Animal
  warm_blooded: yes

Elephant
  subclass:      Mammal
  * colour:      grey
  * size:        large

Clyde
  instance:      Elephant
  colour:        pink
  owner:         Fred

Nellie:
  instance:      Elephant
  size:          small
```

A particular frame (such as Elephant) has a number of *attributes* or *slots* such as `colour` and `size` where these slots may be filled with particular values, such as `grey`. We have used a ``*'' to indicate those attributes that are only true of a *typical* member of the class, and not necessarily every member. Most frame systems will let you distinguish between typical attribute values and definite values that must be true.

[Rich &Knight in fact distinguish between attribute values that are true of the class itself, such as the number of members of that class, and typical attribute values of members]

In the above frame system we would be able to infer that Nellie was small, grey and warm blooded. Clyde is large, pink and warm blooded and owned by Fred. Objects and classes inherit the properties of their parent classes UNLESS they have an individual property value that conflicts with the inherited one.

Inheritance is simple where each object/class has a single parent class, and where slots take single values. If slots may take more than one value it is less clear whether to block inheritance when you have more specific information. For example, if you know that a mammal *has_part* head, and that an elephant *has_part* trunk you may still want to infer that an elephant has a head. It is therefore useful to label slots according to whether they take *single values* or *multiple values*.

If objects/classes have several parent classes (e.g., Clyde is both an elephant and a circus-animal), then you may have to decide which parent to inherit from (maybe elephants are by default wild, but circus animals are by default tame). There are various mechanisms for making this choice, based on choosing the most specific parent class to inherit from.

In general, both slots and slot values may themselves be frames. Allowing slots of be frames means that we can specify various attributes of a slot. We might want to say, for example, that the slot *size* always must take a *single value* of type *size-set* (where size-set is the set of all sizes). The slot *owner* may take *multiple values* of type *person* (Clyde could have more than one owner). We could specify this in the frames:

```
Size:
  instance:        Slot
  single_valued:   yes
  range:           Size-set

Owner:
  instance:        Slot
  single_valued:   no
  range:           Person
```

The attribute value Fred (and even large and grey etc) could be represented as a frame, e.g.,:

```
Fred:
  instance:        Person
  occupation:      Elephant-breeder
```

One final useful feature of frame systems is the ability to attach procedures to slots. So, if we don't know the value of a slot, but know how it could be calculated, we can attach a procedure to be used *if needed*, to compute the value of that slot. Maybe we have slots representing the length and width of an object and sometimes need to know the object's area - we would write a (simple) procedure to calculate it, and put that in place of the slot's value. Such mechanisms of *procedural attachment* are useful, but perhaps should not be overused, or else our nice frame system would consist mainly of just lots of procedures, interacting in an unpredictable fashion.

Frame systems, in all their full glory, are pretty complex and sophisticated things. More details are available in [Rich &Knight, sec 9.2] or [Luger &Stubblefield, sec 9.4.1] (or less detail in [Bratko]). The main idea to get clear is the notion of inheritance and default values. Most of the other features are developed to support inheritance reasoning in a flexible but principled manner. As we saw for nets, it is easy to get confused about what slots and objects really mean. In frame systems we partly get round this by distinguishing between default and definite values, and by allowing users to make slots `first class citizens', giving the slot particular properties by writing a frame-based definition.

Next: Exam-like questions **Up:** Structured Objects **Previous:** Semantic Nets

---

---

alison@
Fri Aug 19 10:42:17 BST 1994