Knowledge Patterns: Working Note 13

Peter Clark
Knowledge Systems
Boeing Applied Research and Technology
peter.e.clark@boeing.com

Dec 8th 1998

Abstract

In our earlier work, components were considered self-contained logical theories, and composition was performed by simply including these theories in a larger KB, with suitable inference to account for interactions among components.

In this working note, we sketch out a more general vision of components, which is more in tune with our original goal that they capture "cliches" or "patterns" of relationships. In this approach, a component's axioms are transformed, or *morphed*, before being imported to a KB, by renaming its symbols. We describe how this can be done by manually specifying such renamings, and then how such renamings can be automatically computed by the inference engine, including for potentially unanticipated situations. Finally, we describe how questions can be answered by run-time construction of a "relevant" KB using this approach, rather than by querying a massive, pre-built KB.

1 Introduction

In our earlier work on "knowledge components", a component was treated as a self-contained theory which could be included in a larger, more detailed representation of some pheonomenon ("the KB"). Concepts in the KB were "mapped" to concepts in the theory using articulation axioms, such as inheritance axioms (eg. "Fuel-Tank ISA Container") or slot-linking axioms ("the MAX-AMOUNT-OF-FUEL of a Fuel-Tank = the CAPACITY of the Container.").

Although this approach provides some useful modularity for KB design and maintenance, it is limited in (at least) two respects, which we discuss in this paper. First, it does not allow representation of *patterns* ie. overly general templates denoting recurring theory schema; instead, a theory's axioms are imported directly into the KB, and thus treated as universally true. This conflicts with our (and others') original thesis that recurring patterns or 'cliches' should be captured and encoded. Second, all mappings between the KB and component theories have to be pre-enumerated by hand by the knowledge engineer: the approach thus does not allow for automatic recognition that a pattern may apply, and automatic application of a pattern. In this working note, we describe an extension of our earlier approach which accounts for both these phenomena.

2 Theories, Patterns, and Morphisms

2.1 Theories and Patterns

We define a pattern as a theory 'template' or 'schema', which denotes a *structure*[1] of objects and relationships, but whose axioms are *not* directly part of the global KB. Rather, a pattern becomes included in the KB through some renaming of its symbols (a *signature morphism*). For example, the axiom:

;;; "If a consumer is connected to a producer, then it is supplied."
$\forall C \; \exists P \;\; consumer(C) \wedge producer(P) \wedge connects(C, P) \rightarrow supplied(C)$

is better viewed as a pattern rather than an axiom, because (for example) we do not intend this to say that being connected to *any* producer ensures supply – rather, the consumer must be connected to a producer *of the same stuff which the consumer consumes*[2]. For example, a light-bulb (a type of consumer) connected to a water-pump (a type of producer) is not supplied in the intended sense.

Note that whether an axiom is a "pattern" or a "true statement" (for direct inclusion in the KB) is not an intrinsic property of the axiom, but is a subjective decision, ie. depends on the relationship between what the axiom asserts, and what the knowledge engineer intended to assert.

A pattern becomes incorporated in the KB via renaming of its non-logical symbols, or *signature*. (We can loosely think of this as "instantiating" the pattern). For example, the above axiom's signature `consumer-producer-connects-supplied` may be transformed ("morphed") to `light-battery-connects-powered` and then the transformed axioms incorporated in the KB. One can think of this process as 'specializing' or 'instantiating' a theory, although strictly it is purely one of transforming a theory, as the theory's symbols (`consumer` etc.) are independent of and have no relationship to the symbols in the target KB (eg. we could just have easily named the theory symbols `s1-s2-s3-s4`).

2.2 Specifying Morphisms: Mapping by Name

Following the terminology of [Falkenhainer et al., 1986], we refer to the source theory as the *base*, and the KB to import it as the *target*. The mapping of symbols from the base theory to the target KB is called a *signature morphism* (or, below, we sometimes write just *morphism*). For example:

```
AXIOM (``pattern''):
   "If a CONSUMER is connected to a PRODUCER, then it is supplied."

(one possible) MORPHISM:
   CONSUMER -> LIGHT
   PRODUCER -> BATTERY

TRANSFORMED AXIOM (for inclusion in the KB):
   "If a LIGHT is connected to a BATTERY, then it is supplied."
```

[1] We say two theories share the same structure if they can be made identical solely through symbol renaming.

[2] (We will ignore other over-generalizations in this example)

Here, the symbols `CONSUMER` etc. become replaced with `LIGHT` etc. before being included in the main KB. (As a shorthand, we do not list elements of the signature which remain untransformed, eg. `connected -> connected`).

In this approach, we have referred to the target objects in the KB by *name* (`LIGHT` etc.). However, there is also a more interesting approach in which we refer to the target objects by description. This provides the potential for the system, rather than the user, to automatically determine how to morph a theory into the KB, possibly in a novel way to answer an unexpected question.

2.3 Specifying Morphisms: Mapping by Description

When importing a theory, the symbols in the theory's axioms are mapped (morphed) to objects in the target KB. In the previous Section, we specified a mophism by giving (one possible set of) names for those objects. However, more generally we can instead give a *description* of those objects: In this example, the `CONSUMER` corresponds to (the class of) things which consume some transport element `T`, and the `PRODUCER` corresponds to (the class of) things which produce the same transport element `T`. In the light-battery case, the transport-element `T` is electricity. For other morphisms, `T` will be different (eg. fuel, water, air, ideas,...).

We can thus say the morphism is *parameterized* by the value of `T`, ie. varying `T` varys the morphism, or similarly providing `T` determines the morphism to apply. Thus, rather than naming sets of target KB symbols (each set = a possible morphism), we can provide a single, parameterized description of them. The identity of these object descriptions can then be found at run-time, ie. the inference engine, rather than the user, can identify the target classes to map the theory's symbols to, by finding classes matching those descriptions.

For example, the earlier axiom expressed in this way would thus look:

```
AXIOM (‘‘pattern’’):
   "If a CONSUMER is connected to a PRODUCER, then it is supplied."

PARAMETER:
   TRANSPORT-ELEMENT

MORPHISM:
   CONSUMER -> (the-class-of THING with (consumes (TRANSPORT-ELEMENT)))
   PRODUCER -> (the-class-of THING with (produces (TRANSPORT-ELEMENT)))
```

Note here the morphism maps the theory symbols to a set of classes, described by the parameterized descriptions shown (`(the-class-of...)`).

Another way to view the pattern + morphism is that it encodes a "parameterized theory", that is, a theory whose assertions depend on the parameter `TRANSPORT-ELEMENT`. Substituting the object descriptions for the object names in the above axiom, it asserts:

> "If a *thing consuming a particular tranport-element* is connected to a *thing producing the same transport-element* then it is supplied."[3]

[3] In a fuller treatment, we should also qualify `connected` and `supplied` by the transport element, ie. the axiom should read "...connected, in the sense of transport element,...supplied with transport-element.". We ignore these extra qualifications here.

The pattern + morphism structure is in effect a way of factoring this longer axiom into (i) a set of concept descriptions, and (ii) a set of axioms about those described concepts.

The advantage of this approach is that the knowledge engineer does not need to pre-classify objects as "producers", "consumers" etc. Rather, that classification can be done automatically and on-demand. For example, if the user suggests that a light is a consumer of electricity, then to apply this theory the system needs to also find the (class of) object mapping to producer: this class can be found as, according to the definitions in the morphism, `PRODUCER` maps to the class of thing which (in this case) produces electricity. The system can thus search the KB for such an object (eg. a battery is a thing which produces electricity), and, if found, can then apply the morphism.

Similarly, suppose a question is asked about a shop being supplied with goods, suggesting this theory might be applicable. To find how to apply the theory, the system needs to locate the (class of) object which could map to `PRODUCER` in this case, namely something which produces those goods. From searching the KB, the system may determine that 'factory' satisfies this description, and thus the morphism `producer-consumer` → `factory-shop` has been determined. Then, the theory can be imported subject to these symbol renamings, and then used to answer the original question. In particular, note that the morphism `producer-consumer` → `factory-shop` did not have to be pre-authored by the knowledge engineer, but is computed at run-time on demand.

3 An Illustration in KM

We now provide illustrations of how this looks in KM, in several stages.

3.1 Mapping by Name

First we define a base theory, expressing a "pattern" or "cliche" about distribution:

```
;;; ========================================================================
;;;             DISTRIBUTION NETWORK THEORY
;;; A trivialest theory of distribution, which we'd like to morph...
;;; ========================================================================

;;; "A consumer is supplied if it is connected with a Producer."
(every Consumer has
  (consumes (Transport-Element))
  (supplied? (((Self connects) includes (a Producer)))))

(every Producer has
  (produces (Transport-Element)))
```

We now reify the concept of a morphism in KM, as an object with a set of mappings from base symbols to some target classes. For example, here we define how to map the distribution network theory's signature `producer-consumer-transportelement` onto `battery-light-electricity`:

```
KM> (a Morphism with
        (mappings (
            (a Mapping with (base (Producer)) (target (Battery)))
            (a Mapping with (base (Consumer)) (target (Light)))
            (a Mapping with (base (Transport-Element)) (target (*Electricity))))))
_Morphism1
```

We now extend KM's `(load-kb...)` mechanism to also include the ability to rename symbols[4], by allowing a `:with-morphism` option, eg:

```
KM> (load-kb "distribution.km" :with-morphism _Morphism1)
```

This directive does not load the file distribution.km verbatim, but morphs it as specified by `_Morphism1`. The morphed theory, now in the KB, looks:

```
KM> (write-kb)
(every Light has
  (consumes (*Electricity))
  (supplied? (((Self connects) includes (a Battery)))))

(every Battery has
  (produces (*Electricity)))
```

3.2 Mapping by Description

We can similarly provide *descriptions* for the target concepts in a morphism. To do this, we add a minor extension to KM to find the *class* matching a given description, by introducing a `the-class-of` directive, eg.

```
KM> (every Battery has
        (produces (*Electricity)))

KM> (the-class-of Thing with (produces (*Electricity)))
Battery
```

This latter directive finds the most general class(es) which "match" (ie. are subsumed by) the given description.

Now, rather than creating individual morphisms for every possible type of transport element, we can create a single, parameterized morphism:

```
KM> (a Morphism with
        (mappings (
            (a Mapping with
              (base (Producer))
              (target ((the-class-of Thing with
                                (produces ((the transport-element of Self)))))))
            (a Mapping with
              (base (Consumer))
              (target ((the-class-of Thing with
                                (consumes ((the transport-element of Self)))))))
            (a Mapping with
              (base (Transport-Element))
              (target ((the transport-element of Self)))))))
_Morphism2
```

The `the-class-of` expressions in the morphism here appeal to the (as yet unspecified) value of the transport-element slot, ie. the morphism is parameterized by this slot's value. We can provide a value for this parameter, then have KM resolve the target descriptions for this value:

[4] In fact, the Lisp function `sublis` does exactly this, making the implementation trivial.

```
;;; Set the morphism's parameter...
KM> (_Morphism2 has (transport-element (*Electricity)))

;;; Print out the morphism...
KM> (forall (the mappings of _Morphism2
            (:set (the base of It) (the target of It)))
(Producer Battery                              ; ie. Producer maps to Battery
 Consumer Light                                ; ie. Consumer maps to Light
 Transport-Element *Electricity)               ; ie. Transport-Element maps to *Electricity
```

And again load in the base theory using this morphism:

```
KM> (load-kb "distribution.km" :with-morphism _Morphism2)
```

3.3 Reifying the Components in the KB

To integrate this together, we now reify our base theory, ie. create a frame to denote it and its properties in the KB. This frame describes the theory's source file, and the parameterized morphism required to import it:

```
KM> (Distn-Theory has (superclasses (Theory)))

KM> (every Distn-Theory has
     (source-file ("distribution.km"))          ; file containing the axioms
     (transport-element ())                     ; A parameter of the theory
     (morphism ((a Morphism with                ; the morphism to apply
        (mappings (
            (a Mapping with
              (base (Producer))
              (target ((the-class-of Thing with
                                (produces ((the transport-element of Self)))))))
            (a Mapping with
              (base (Consumer))
              (target ((the-class-of Thing with
                                (consumes ((the transport-element of Self)))))))
            (a Mapping with
              (base (Transport-Element))
              (target ((Self transport-element)))))))))
```

Note that we represent the distribution theory as a class, not an instance, as strictly it represents a *collection* theories in the target KB, each one corresponding to a different value of the `transport-element` parameter. Different instances of `Distn-Theory` will have different values for the `transport-element` slot, and hence different morphisms.

We can now define a KM command `(import` *theory-instance*`)` import a theory:

```
KM> (import (a Distn-Theory with (transport-element (*Electricity))))
```

This command causes KM to

(i) compute the morphism for this theory instance (ie. find the value of the `morphism` slot), and then

(ii) do a `load-kb` on this theory, applying that morphism in the process.

3.4 Demand-Driven Importing of Theories

So far, we have described how 'patterns' can be viewed as parameterized theories, and how KM can itself compute the appropriate morphism to use to import a theory, given the value(s) of those parameter(s).

We now show how theory importing can be done in a demand-driven way, simply by placing the `import` directive on a frame's slot. Then, if ever the value of that slot is requested, KM will evaluate the `import` expression and thus import the named theory, morphing it in the appropriate way. This may include creating novel morphisms, unanticipated by the knowledge engineer.

For example, we could place the import directive on a `Consumer` frame in the target KB:

```
(every Consumer has
  (supplied? ((import (a Distn-Theory with
                               (transport-element ((the consumes of Self))))))))
```

Now suppose a `Light` has been declared as an electrical consumer in the target KB, eg:

```
(Light has (superclasses (Consumer)))

(every Light has (consumes (*Electricity)))
```

Now suppose we ask "Is Light1 supplied [with electricity]?". Light1 inherits the import expression on the `supplied?` slot on `Consumer`, which will then be evaluated by KM. This causes the distribution theory to imported, morphed with the transport-element parameter being set to `*Electricity`. Thus, the KB acquires axioms like "If the Light is connected to a Battery, then it is supplied.", and now the "supplied?" question can be answered using these new axioms.

We need not have explicitly declared `Light` as a `Consumer`; as an alternative, we could have placed the `supplied?` slot on `Thing` instead, so that if ever a query asks whether something is supplied, importing of the "distribution network" theory is triggered, morphed appropriately.

As another example, suppose we now ask if a person is being supplied with news. Again, this triggers importing of the distribution network theory, morphed this time with a `transport-element` of news information, thus requiring KM to find, as the target symbol for `Producer`:

```
(the-class-of Thing with (produces (*News-Information)))
```

KM may thus locate `Newspaper` as the class matching this description. Hence one of the imported axioms this time will be (say):

```
(every Person has
  (supplied? (((Self connects) includes (a Newpaper)))))
```

For a fuller treatement, we should also morph `supplied?` and `connects` to be supplied *in the sense of news information* and connects *in the sense of news information*, eg. `is-informed` and `has-access-to` – this just requires adding additional mappings to the morphism. We ignore this here for simplicity.

4 Summary and Reflection

We now reflect on the questions of: What is going on here, and why is it significant?

What is going on? First, in this approach, a component theory is not a partition of the knowledge base. Rather, it is a cliche, or pattern, denoting a whole family of theories which share the same structure. It truely achieves our aim of capturing those recurring patterns which get applied in a miriad of ways to model real-world phenomena.

Second, we have shown that rather than manually specifying all the possible morphisms of the theory to the target KB by *name*, we can instead specify the target objects by a single *parameterized description*. The morphism to apply can thus be computed automatically at run-time, given just the value of those parameters.

Finally, we have shown that by placing the command to import a theory on a frame's slot, KM can morph and load theories in a demand-driven way at run-time.

This approach is potentially significant in several ways, although these are largely conjecture at present. First, we conjecture these 'pattern' theories are more reusable, as they strip out the many possible domain-specific qualifications to their assertions (or more precisely, state those qualifications in the description of how to map the theory to the target KB). Second, as the target objects for the mappings are specified by description, rather than by name, the inference engine itself can identify those targets, including for situations unanticipated by the knowledge engineer. Third, the system can identify, morph, and import relevant theories on demand to answer questions, thus controlling the size of the actual KB used for inferencing – this contrasts with the earlier approach, where all component theories were assembled into a giant KB pre-query-time. This controlled creation and assembly of a "relevant" composition for question-answering seems essential for problem-solving in broad scope applications, where working with a single, giant KB is impractical.

References

[Falkenhainer et al., 1986] Falkenhainer, B., Forbus, K. D., and Gentner, D. (1986). The structure-mapping engine. In *AAAI-86*, pages 272–277.