More on Components: Working Note 12

Peter Clark
Boeing Research
PO Box 3707, Seattle, WA 98124
peter.e.clark@boeing.com

Bruce Porter
Department of Computer Sciences
University of Texas at Austin, TX 78712
porter@cs.utexas.edu

Feb 27th 1998

Abstract

This working note aims to develop the ideas of our earlier AAAI'97 paper [Clark and Porter, 1997] into a more coherent implementation framework. We would like to reach the stage where building and linking components becomes a routine task, but this requires a concrete description of what a component data structure looks like and how it would be used. This document aims to do this, adding in some important specifics to the earlier work.

1 Introduction

This working note aims to develop the ideas of our earlier AAAI'97 paper [Clark and Porter, 1997] into a more coherent implementation framework. We would like to reach the stage where building and linking components becomes a routine task, but this requires a concrete description of what a component data structure looks like and how it would be used. This document aims to do this, adding in some important specifics to the earlier work.

2 Components

2.1 Architecture and Usage

The idea of a component is to encapsulate a 'coherent' mini-theory, which may be incorporated (perhaps through several different mappings, described later) in a knowledge base (KB). The goal is to save the knowledge engineer trying to build the 'mother of all representations' when building a KB, ie. trying to represent every conceivable nuiance about the domain of interest – rather he/she encodes a number of small, coherent modules (components), which can then be composed together to build a KB.

It is important to distinguish two separate tasks:

Specification: *Specifying* how one component is a composition of others. This is a task which the knowledge engineer does.

Composition: *Building* the composition, ie. 'merging' components together in the way which the knowledge enginer specified. This is a task which the computer does automatically using the specifications.

Each component contributes axioms (assertions) to a composition. A composition is thus a larger set of axioms, built subject to the mappings which the user specified.

Note that there is no longer the notion of a single, universal KB. Rather, each component can be thought of as specifying a 'mini KB', which can be assembled my merging the components it imports. If this seems strange, and you would prefer to still think in terms of building a single KB, then arrange the components such that there is one bottom-most component, build from all the others, which can be thought of as the Target KB to build.

Note also we're here using a slightly different notion of composition than in the AAAI'97 paper. Here, composition means simply 'assembling the axiom set', and doesn't include computing ramifications of that assembly (the latter is done at run-time in response to user questions). Then, run-time question answering is performed by posing questions to this set. In our particular implementation, we choose to also first load this axiom set into a conventional Knowledge Representation System (here KM), ie. a KM KB is built from the axiom set. This extra step can be thought of as "compiling" the axiom set for efficient inference. Thus note that we are not proposing a new representation language, but rather a more reusable way for constructing KBs in existing languages (eg. KM, Algernon)

2.2 Anatomy of a Component

A component consists of 6 items:

`:name` A identifier for the component.

`:description` A string of text describing it.

`:participants` A set of objects which the component is describing.

`:axioms` A set of axioms (assertions) about the participants.

`:import-components` A set of components which this component imports.

`:parameters` Sometimes, it is desirable to parameterize the participants of a component.

The semantics of a component are, informally, as follows: For all instances of that component, there exist instances of the participants such that all the axioms hold.

A simple example of a component, which does not import any other components, is the one below. This states that a hydraulic circuit consists of a pump and an actuator, that the pump powers the actuator, and that the pump's output equals the actuator's input:

```
(defcomponent
  :name Hydraulic-circuit
  :description "A trivial model of a hydraulic circuit"
  :participants
        P:Pump
        A:Actuator
  :axioms
        powers(P,A)
        output(P,O) :- input(A,O)
)
```

with the semantics that:

```
forall X:hydraulic-circuit
exists
     P:Pump
     A:Actuator
such that
     powers(P,A)
and  forall O   output(P,O) <- input(A,O)
```

In general, however, a component will also import other components. If a component is imported, then the system must

- work out which participants in the imported component correspond to participants in the importing component,
- unify those matching participants, and
- union the resulting axioms together.

We call this first step (working out the mapping between participants) as *aligning* the participants of the two components. There are several ways in which this can be implemented. In the AAAI'97 paper, we suggested that the knowledge engineer manually specify the mapping by placing 'role' labels on each participant. However, in retrospect this seems likely to be painful for components with very large numbers of participants)[1].

Another alternative, which we adopt here (though it's by no means the best), is to use a unification algorithm to automatically determine which participant in one component will match with which participant in the other component. KM already has such a mechanism built in (see the `&&` operator in the KM User Manual [Clark and Porter, 1996]). For example, using the `&&` operator, KM will unify the participant set {`_pump1 _actuator2`} from one component with the participant set from another component {`_edp3 _wing4`} to produce the unified set {`_edp3 _actuator2 _wing4`} (where `_pump1` is a pump, `_actuator2` is an actuator, `_edp3` is an engine-driven pump, and `_wing4` is a wing). Here, KM is using it's knowledge about the classes of the objects being merged to find 'compatible' matches. Note that not all participants in one component need to have a corresponding participant in the other – thus an imported component may introduce extra participants into the importing component. This unification-based approach to 'aligning' participants is rather pragmatic, and has some open issues to resolve, but it's the best we can think of for now. One interesting aspect is that there may be more than one way in which two participant sets can be aligned; in this case, the component can be applied in more than one way, corresponding to viewing an object in more than one way. In this case, some additional mechanism would be needed so that the knowledge engineer can specify which alignment she/he intended.

Figure 1 sketches the 'imports' relationship that might exist between components, based on the example described in more detail later. Figure 2 shows the composition being computed, including aligning the participants from different components together. The black dots denote participants explicitly declared in the local component, the white dots denote participants imported from other components. This figure also suggests that the final axiom set may be compiled into a more standard representational form (eg. a KM KB) for question-answering.

The algorithm for building a composition is shown in Figure 3. The final axiom set can then be asserted into a KM KB using KM's `also-has` primitive.

[1]Another problem is as follows: Consider a component C importing two components C1 and C2: A participant in one imported component C1 may map to a participant in the other imported component C2, without that participant ever being mentioned in the importing component C! So do we create extra participants in C for all potentially imported participants?

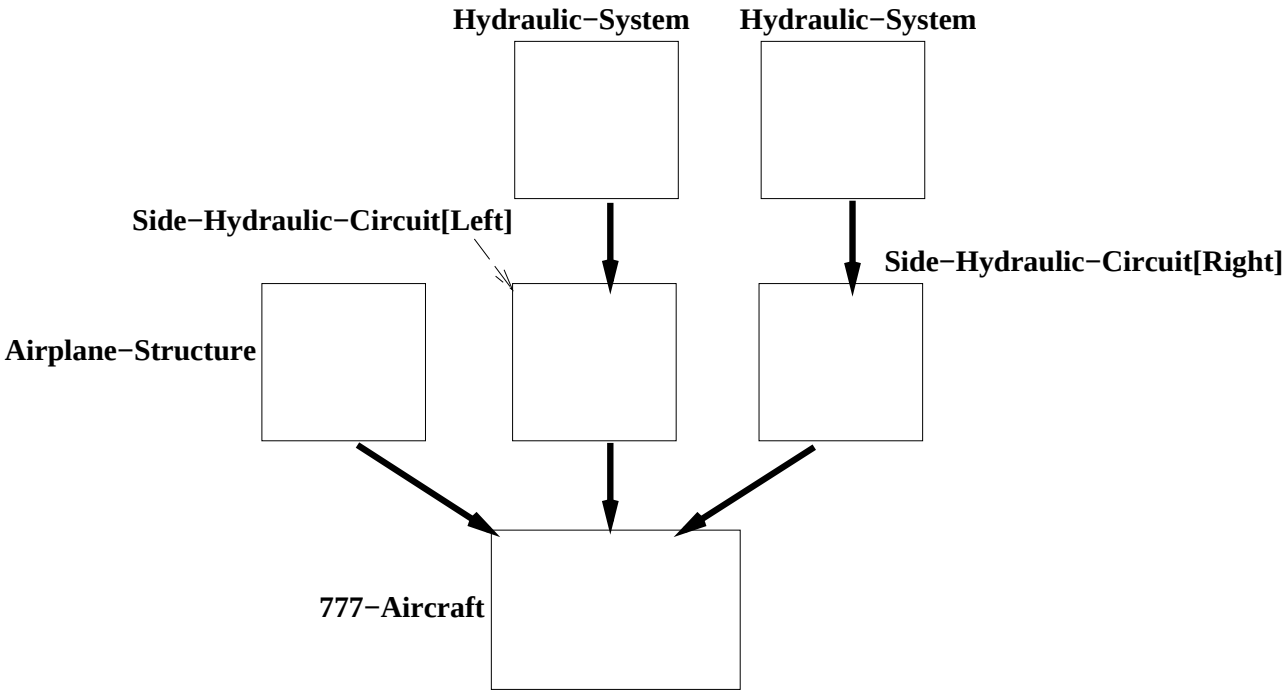


Figure 1: Example of 'imports' relationships between components.

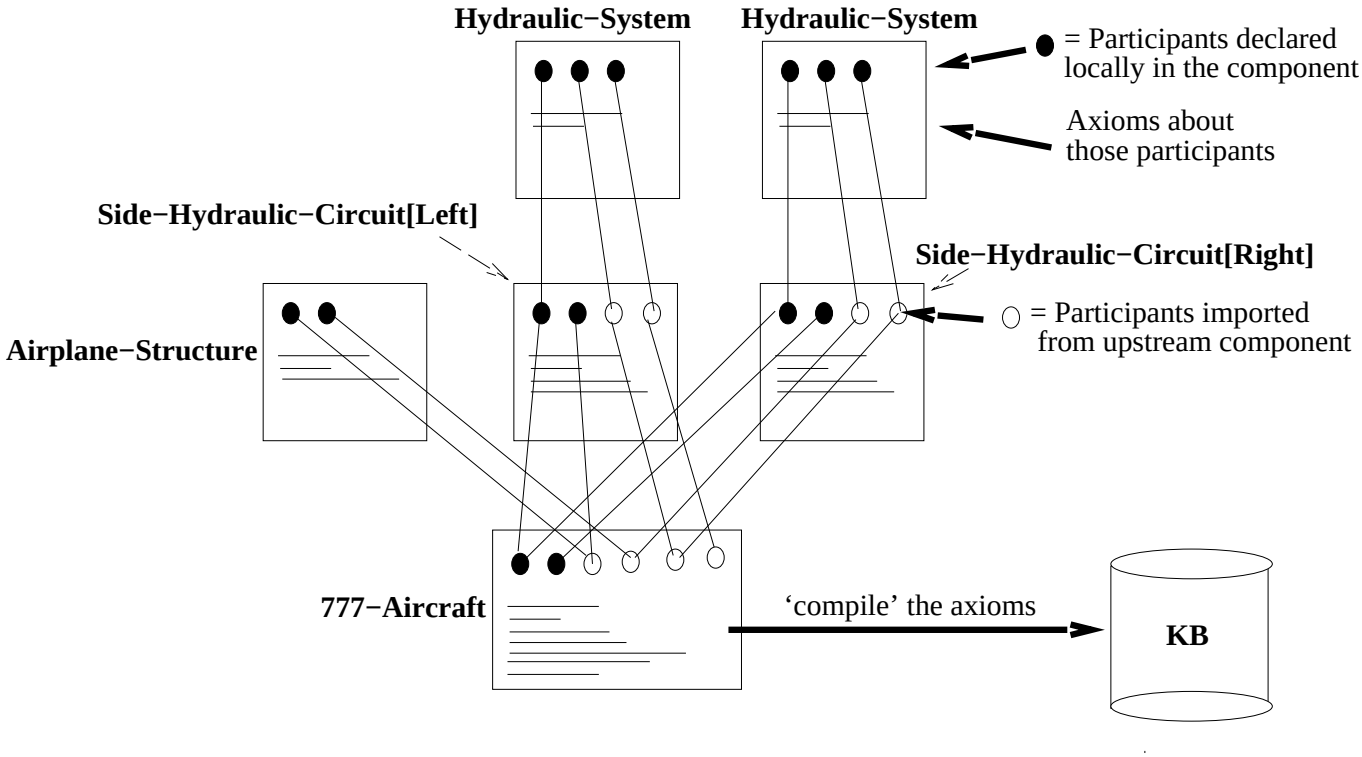


Figure 2: Composition of the components, showing how the participants in each component have been (automatically) aligned. The final assembly of axioms can be 'compiled' (asserted) into a standard knowledge-base.

```
PROCEDURE Build-Component (Component-Name) returning {Participants,Axioms}:
  1. Create instances denoting the participants P in the component
  2. For each imported component:
        2a. Call Build-Component to find it's participants P' and axioms A'
        2b. unify sets P and P'
        2c. Add axioms A' to the component's local set A
  3. Return the final set of participants P and axioms A
END-PROCEDURE

PROCEDURE Build-KB (Component)
  1. Call Build-Component(Component) returning Axioms
  2. For each Axiom in Axioms
       - assert Axiom into the KB
END-PROCEDUREA
```

Figure 3: Composition algorithm.

3 Example

The following pages show four simple examples of components which connect together to build a toy KB of an airplane. The last component (`777-Aircraft`) can be thought of as "The KB". The result of assembling it (by importing other components and aligning their participants) is also shown.

Although most of the axioms shown for a component are ground clauses, they can be any axiom at all – A few non-ground ones are included also for illustration purposes.

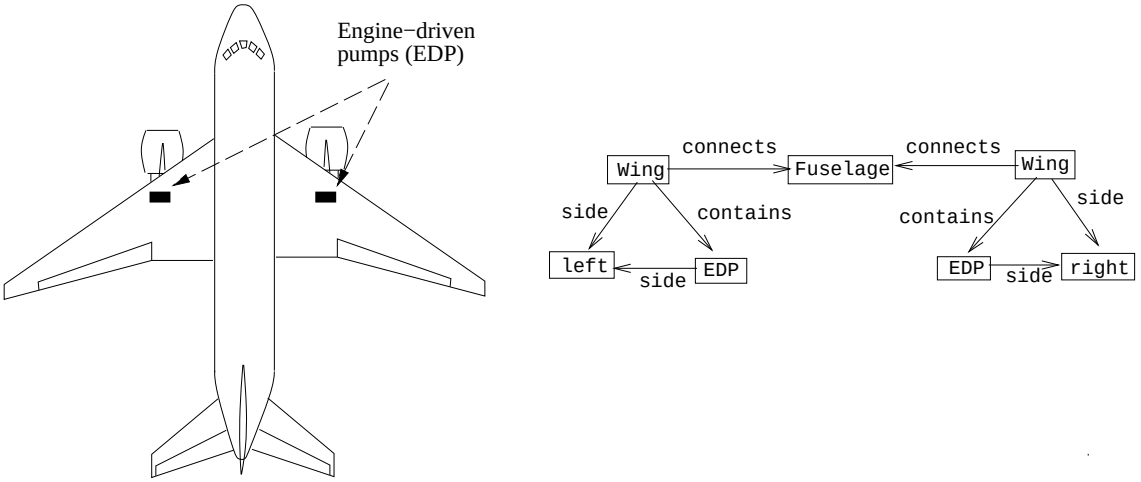
The KM representation of the components is rather clunky (although semantically correct); it will have to do for now. Axioms are expressed as *frame-slot-value* triples, where *value* may be an arbitrarily complex KM expression. A triple is denoted in KM using the notation `(:triple` *frame slot val-expr*`)`. Each triple in the final axiom set can be asserted into a KM KB using the standard KM notation `(`*frame* `also-has (`*slot* `(`*val-expr*`)))`.

Component: Airplane-Structure

Synopsis

Name: `Airplane-Structure`
Summary: Basic physical layout of an aircraft
Parameters: (none)
Imported components: (none)

Description



This component describes some of the physical structure of the aircraft, in particular focussing on hydraulic components (the engine-driven pump).

Local Participants (English)

- The left wing of the airplane
- The right wing
- The left engine-driven pump (EDP)
- The right EDP
- The fuselage

Local Axioms (English)

- The left wing is connected to the fuselage
- The right wing is connected to the fuselage
- The left wing contains the left EDP
- The left wing contains the right EDP

Component representation (semi-formal)

```
(defcomponent
  :name Airplane-Structure
  :description "Structure of a 777 aircraft"
  :participants
     LW:Wing where side(LW,left)
     RW:Wing where side(RW,right)
```

```
    LE:EDP  where side(LE,left)
    RE:EDP  where side(RE,right)
    F: Fuselage
  :axioms
    connects(LW,F)
    connects(RW,F)
    contains(LW,LE)
    contains(RW,RE)
)
```

Component representation (KM notation)

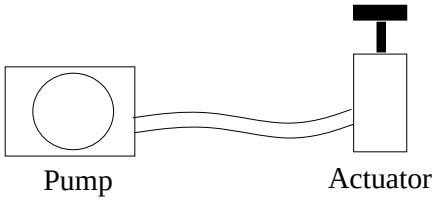
```
(defcomponent
  :name Airplane-Structure
  :description "Structure of a 777 aircraft"
  :participants (
    (a Wing with (side (Left)))
    (a Wing with (side (Right)))
    (a Edp with (side (Left)))
    (a Edp with (side (Right)))
    (a Fuselage))
  :axioms (
    (:triple (allof ((Self participants Wing)) where ((It side) = Left))
                    connects (Self participants Fuselage) )
    (:triple (allof ((Self participants Wing)) where ((It side) = Right))
                    connects (Self participants Fuselage) )
    (:triple (allof ((Self participants Wing)) where ((It side) = Left))
                    contains
             (allof ((Self participants EDP)) where ((It side) = Left)))
    (:triple (allof ((Self participants Wing)) where ((It side) = Right))
                    contains
             (allof ((Self participants EDP)) where ((It side) = Right))))
)
```

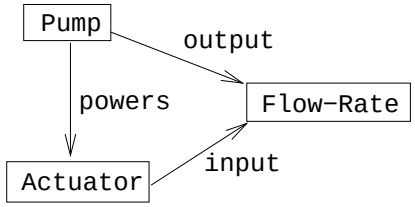
Component: Hydraulic-Circuit

Synopsis

Name: `Hydraulic-Circuit`
Summary: Generic hydraulic circuit
Parameters: (none)
Imported components: (none)

Description





Describes the relationship between a pump and an actuator. There are axioms here describing the power transfer relationship (namely, the pump powers the actuator), and constraints on the flow-rate. In a fuller implementation, these would more likely be two separate, generalized components (one for power transfer between a producer and consumer, and one for fluid-flow).

Local Participants (English)

- A pump
- An actuator

Local Axioms (English)

- The pump powers the actuator
- The output flow-rate of the pump equals the input flow-rate to the actuator

Component representation (semi-formal)

```
(defcomponent
  :name Hydraulic-circuit
  :participants
        P:Pump
        A:Actuator
  :axioms
        powers(P,A)
        forall O  output(P,O) :- input(A,O)
)
```

Component representation (KM notation)

```
(defcomponent
  :name Hydraulic-circuit
  :participants (
        (a Pump)
```

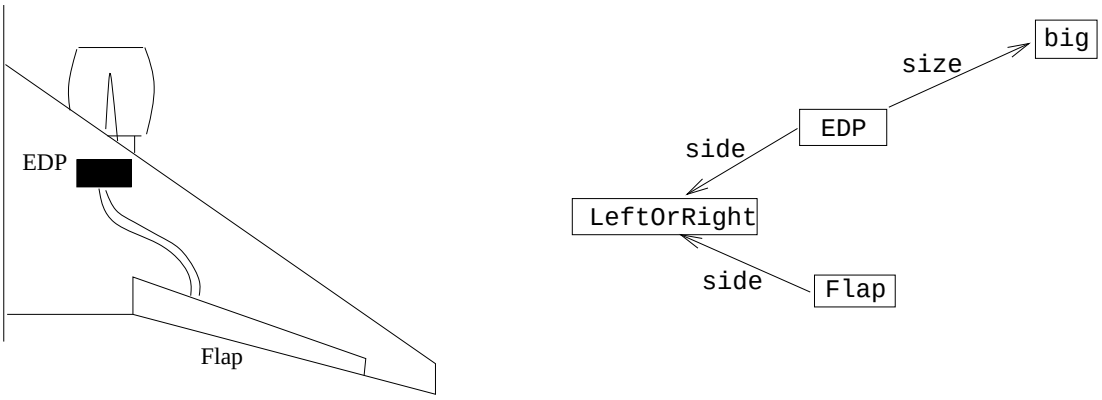
```
        (a Actuator))
  :axioms (
        (:triple (Self participants Pump) powers (Self participants Actuator))
        (:triple (Self participants Pump) output (Self participants Actuator input))
)
```

Component: Side-Hydraulic-Circuit

Synopsis

Name: `Side-Hydraulic-Circuit`
Summary: The side hydraulic circuit of a 777 aircraft
Parameters: `side` (one of {`Left`,`Right`})
Imported components: `Hydraulic-Circuit`

Description



This component describes the presence of a hydraulic circuit on an aircraft. The imported `Hydraulic-circuit` component provides the generic axioms about hydraulic circuits, while this component maps those participants into participants of the airplane, and adds (purely for demo purposes) the axiom that the engine-driven pump is big.

Local Participants (English)

- An engine-driven pump (EDP)
- A flap

Local Axioms (English)

- The EDP is big

Component representation (semi-formal)

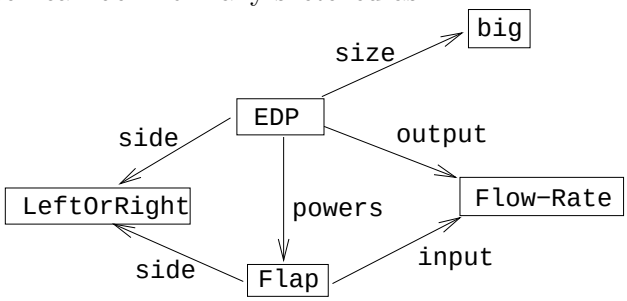
```
(defcomponent
  :name side-hydraulic-circuit
  :description "The side hydraulic circuit"
  :import-components (Hydraulic-circuit)
  :parameters
        side=W:LeftOrRight
  :participants (
        E:EDP  where side(E,W)
        F:Flap where side(F,W)
  :axioms (
        size(E,big)
)
```

Component representation (KM notation)

```
(defcomponent
  :name side-hydraulic-circuit
  :description "The side hydraulic circuit"
  :import-components (Hydraulic-circuit)
  :parameters (
        (side ((a Side))))
  :participants (
        (a Edp with (side ((Self side))))
        (a Flap with (side ((Self side)))))
  :axioms (
        (:triple (Self participants Edp) size Big))
)
```

The Composition

The composition itself can be informally sketched as:



or described as follows:

```
forall S:Side-Hydraulic-Circuit
exists
     E: EDP
     F: Flap
such that
     participant(E,S)
     participant(F,S)
     size(E,big)                          ; local axiom
     powers(E,F)                          ; imported from Hydraulic-Circuit
     forall O output(E,O) :- input(F,O)   ; imported from Hydraulic-Circuit
     side(E,X) :- side(S,X)               ; local, parameterized property of E
     side(F,X) :- side(S,X)               ; local, parameterized property of F
```

Note the two axioms importred and specialized from `Hydraulic-System`, stating:

- The EDP powers the flap
- The output of the EDP equals the input of the flap

Component: 777-Aircraft

Synopsis

Name: `777-Aircraft`
Summary: The whole 777-aircraft
Parameters: (none)
Imported components: `Airplane-Structure`, `Side-Hydraulic-Circuit[Left]`, `Side-Hydraulic-Circuit[Right]`

Description

This component is purely a composition of three other components. It doesn't introduce any new participants or axioms. Here we want to compose knowledge about the aircraft's structure and hydraulic circuits to build a composite representation

Local Participants (English)

(none)

Local Axioms (English)

(none)

Component representation (semi-formal)

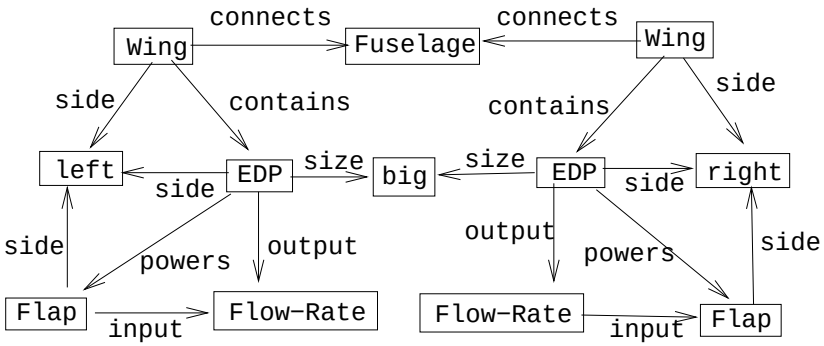
```
(defcomponent
  :name 777-aircraft
  :description "The whole 777 aircraft, just looking at hydraulics"
  :import-components (
        Airplane-structure
        Side-hydraulic-circuit[side=left]
        Side-hydraulic-circuit[side=right]
)
```

Component representation (KM notation)

```
(defcomponent
  :name 777-aircraft
  :description "The whole 777 aircraft, just looking at hydraulics"
  :import-components (
        (a airplane-structure)
        (a side-hydraulic-circuit with (side (Left)))
        (a side-hydraulic-circuit with (side (Right))))
)
```

4 Composition

Building the fourth component here using the algorithm described earlier (Figure 3) produces a set of axioms, which can be informally sketched and more formally expressed as follows:




```
forall A:777-aircraft
exists
     LW:Wing
     RW:Wing
     LE:EDP
     RE:EDP
     LF:Flap
     RF:Flap
     F: Fuselage
such that
     participant(LW,A)
     participant(RW,A)
     participant(LE,A)
     participant(RE,A)
     participant(LF,A)
     participant(RF,A)
     participant(F,A)
     side(LW,left)                ; imported from Airplane-Structure
     side(LE,left)                ; imported from Airplane-Strucuture &
                                  ;     Side-Hydraulic-Circuit[left]
     side(LF,left)                ; imported from Side-Hydraulic-Circuit[left]
     side(RW,right)               ; imported from Airplane-Structure
     side(RE,right)               ; imported from Airplane-Strucuture &
                                  ;     Side-Hydraulic-Circuit[right]
     side(RF,right)               ; imported from Side-Hydraulic-Circuit[right]
     connects(LW,F)               ; imported from Airplane-Structure
     connects(RW,F)               ; imported from Airplane-Structure
     contains(LW,LE)              ; imported from Airplane-Structure
     contains(RW,RE)              ; imported from Airplane-Structure
     powers(LE,LF)                ; imported from Hydraulic-Circuit via
                                  ;     Side-Hydraulic-Circuit[left]
     forall O1  output(LE,O1) :- input(LF,O1) ; from Hydraulic-Circuit via
                                  ;     Side-Hydraulic-Circuit[left]
     powers(RE,RF)                ; imported from Hydraulic-Circuit via
                                  ;     Side-Hydraulic-Circuit[right]
     forall O2  output(RE,O2) :- input(RF,O2) ; from Hydraulic-Circuit via
```

```
                                    ;     Side-Hydraulic-Circuit[right]
    size(LE,big)                    ; imported from Side-Hydraulic-Circuit[left]
    size(RE,big)                    ; imported from Side-Hydraulic-Circuit[right]
```

Note that the component `Side-Hydraulic-Circuit` (and thus `Hydraulic-Circuit`) has been imported twice, once parameterized with `side=left`, and once with `side=right`, corresponding to the two different sides of the airplane.

These axioms may then be subsequently asserted in a standard KM KB for question-answering purposes. Question-answering may of course result in inferencing with these axioms, including normal inheritance and classification behaviour.

5 Additional Issues and Comments for Discussion

5.1 Components and Inheritance

The use of components does not remove the need for normal inheritance-type reasoning and an isa-hierarchy; rather components are an additional mechanism. The underlying model here is that there is already an isa-hierarchy of concepts, and some of those concepts already have axioms (ie. slots + value-expressions) attached to them. The components then make additional assertions to the KB about interactions which exist between set of concepts.

So when should an axiom be placed on a frame in the initial KB, and when should it go in a component? The rule is: if an axiom describes an *intrinsic* (always true) property of an object, then it should go directly into the KB. If, however, it describes an object's property based on it playing a role in some (conceptual) system, then it should go in a component. These latter properties are those which do *not* hold for all instances of that object type, but only those instances participating in that system.

For example, the axioms that an engine-driven pump (EDP) has weight 2.5kg (say) and is made-of titanium (say) are (ie. will be modeled as) intrinsic properties of an EDP – it is always the case that an EDP's weight is 2.5kg, and thus that fact would be placed on the `EDP` frame in the KB taxonomy. However, an axiom stating that the EDP is connected to an airplane's engine is (here modeled as) not universally true (for example, some EDPs are in store-rooms prior to aircraft assembly, not connected to anything), but only true for EDPs which in are part of the wing assembly – in conceptual terms, just those EDPs participating in the *system of relationships* describing the wing assembly's physical structure. This latter axiom would thus be placed on a component for (say) "the physical structure of a wing assembly".

In fact, a standard frame in a KB can be thought of a component as consisting of a single participant, eg:

```
(defcomponent
   :name Pump
   :participants
       P:Pump
   :axioms
       weight(P,2.5)
       material(P,titanium)
)
```

which is then imported into every other component which mentions a participant of the same type. In fact, this is, functionally equivalent to what inheritance does anyway!

5.2 Questioning the KB, and Question-Specific Composition

It seems that questions are often posed in context, for example "*In the wing,* what is the EDP connected to?" or "*In a 777,* what provides emergency hydraulic power?". Earlier, we suggested that a single KB might be built from some "bottom-most component" in the component library. In fact, it seems more approprate that a question-specific mini-KB would be synthesized for each question posed. A component can be thought of as axiomizing objects which exist in a particular context, so given a particular context (stated in a question) the system can build that axiomatization (by building the axiom-set which the component specifies) and then reason with it to generate an answer to that question. Such an axiomatization augments the "base KB" of "universally true" axioms, describing intrinsic properties of objects in the world.

5.3 Some Issues

5.3.1 Aligning Participants

The unification approach to 'aligning' participants may sometimes be ambiguous: For example, a component describing `Service` might have two `Agent`s as participants (one playing the role of a client and one of a server). When importing this component to another also containing two agents, there's ambiguity about which agent should match with which.

In this case, it seems that it would be useful to add role tags to participants so that the knowledge engineer can specify the mapping. A simple implementation trick to do this, without requiring new syntactic extensions, is to introduce a `role` slot to allow the knowledge engineer to tag participants. As the unification algorithm won't allow objects with incompatible slot-values to unify, this would guide the unification algorithm appropriately. For example:

```
(defcomponent
   :name Service
   :participants
       (a Agent with (role (Client)))
       (a Agent with (role (Server)))
   :axioms
        ... )

(defcomponent
   :name Restaurant-Visit
   :import-components (Service)
   :participants
      (a Agent with (role (Client)))
      (a Agent with (role (Server)))
   :axioms
      ...the first Agent is hungry...
      ...the second Agent works for the restaurant...
      etc.)
```

In this example, when the `Service` component is imported into the `Restaurant-Visit` component and the participants are unified, the agent with role `Server` in the `Restaurant-Visit` will only unify with the agent with role `Server`, due to these role tags attached to the agents. This works, but still seems rather a hack – one cause of the root problem here

is that using unification to aligning participants ignores the `:axioms` in the components – in practice, those axioms should play a role in deciding the alignment. As the method currently stands, the system may choose an apparently valid alignment of participants, only to later find that the axioms from the imported component contradicts those in the importing component. There is no mechanism implemented to recover from this in the current implementation (eg. by backtracking to try a different alignment).

5.3.2 Clunky Syntax

The KM syntax in the KM formulations shown in this paper is very clunky – it would be much nicer to move to something more standardized. For example, I was struck how easy it was to download the 20,000 concept WordNet ontology [Miller et al., 1993] from the Web, which is expressed in Prolog, and start doing useful things with it in just a few minutes.

5.3.3 Relevance and Consistency

In this implementation, the composition algorithm builds a component's axiom set in all its detail. An advanced topic would be to have some more sophisticated composition algorithm which would allow some of it's imported components to be selectively left out, on the basis that they were irrelevant to the question being posed.

A second advanced topic would be to have, perhaps, multiple alternative components for describing a particular phenomenon. Again, an algorithm would determine which component was most suitable for answering the current query, and ensure that a composition was performed only with components based on non-conflicting assumptions. This approach has already been demonstrated in compositional modeling of physical systems [Levy, 1993].

6 Methodology for Progress

Consider taking some on-line documentation that we would like to represent (eg. the 777 hydraulics manual). Rather than trying to build the "mother of all KBs" to represent everything, we can instead follow the conceptual organization which the documentation provides, where, approximately, each page will be represented by one component. First we turn to page 1; it provides (say) a description of the physical layout of the aircraft, though biased towards hydraulics – many (but not all) hydraulic elements are shown, while other major structural features (eg. the space for the passengers) are not. A component is built to represent *just the information conveyed on that page*, the physical parts being the component's participants, and the relationships between the parts being the component's axioms. The fact that this, on its own, is a poor description of the whole aircraft does not matter at this stage.

Now we turn to page 2 in our imaginary exercise. It provides (say) a simple overview of the hydraulic circuits. This information is represented as a second component; some of its axioms provide additional information about participants in the first component, others provide information about new participants not previously mentioned. We continue likewise for subsequent pages.

Finally, by building the composition of the components we can merge all this information together to build "the KB". Hopefully, the modularity that this approach provides

will be substantially beneficial for maintaining and further developing such a KB. Time will tell...

References

[Clark and Porter, 1996] Clark,
P. and Porter, B. (1996). KM – the knowledge machine: Users manual. AI Lab, Univ Texas at Austin. http://www.cs.utexas.edu/users/mfkb/manuals/userman.ps).

[Clark and Porter, 1997] Clark, P. and Porter, B. (1997). Building concept representations from reusable components. In *AAAI-97*, pages 369–376, CA. AAAI. (http://www.cs.utexas.edu/users/pclark/papers/aaai97.ps).

[Levy, 1993] Levy, A. Y. (1993). Irrelevance reasoning in knowledge-based systems. Tech report STAN-CS-93-1482 (also KSL-93-58), Dept CS, Stanford Univ., CA. (Chapter 7).

[Miller et al., 1993] Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1993). *Five Papers on WordNet*. Prinston Univ., NJ. (http://www.cogsci.princeton.edu/∼wn/).