Object Descriptions Revisited: Working Note 7

Peter Clark and Bruce Porter
Dept. CS, UT Austin
{pclark,porter}@cs.utexas.edu

1 Introduction

This note revisits the issue of constructing *object descriptions* – namely static descriptions of objects and their relationships (ie. excluding "meta-descriptions" such as actions and scripts). We represent object descriptions using conceptual graphs [Sowa, 1984], ie. semantic networks of nodes (concepts) and arcs (relationships). Our interest is in being able to construct such representations from components, so as to import and apply general knowledge to specific object descriptions automatically.

In the earlier Working Notes 3 and 4, object descriptions were constructed by applying rules or axioms to some initial conceptual graph (representing the object of interest). The axioms elaborated and specialized that initial graph. In those notes, we referred to axioms as the 'components' from which object descriptions were constructed. However, this notion of axioms as components is not completely satisfying:

1. Intuitively (and for practical purposes with a large knowledge base), a single axiom seems too fine-grained to constitute a component. Instead, a component should represent some larger system of inter-relationships (eg. representing *containership*, or *security*, or *transportation*), in which coherent collections of axioms about the same subject are grouped together.

2. In order to apply axioms, we had to hard-wire appropriate `isa` relationships in the isa hierarchy. For example, to apply `X isa container` $\Rightarrow$ `X has portal` to a person (in the context of eating, say) we have to unintuitively put `person isa container` in the taxonomy. Instead, we would like to say that a person *can be viewed as* a container, and appropriately superimpose a model of containership on a person when needed.

In this note, we describe a different notion of object-description component, in which a component represents a *system* of concepts, with inter-relationships described by a *set* of axioms. A component can be added to an object-description by stating how that component represents a *view* of concepts in that description. The definition of a component here follows the GenVoca model presented in detail in the previous Working Note (number 6).

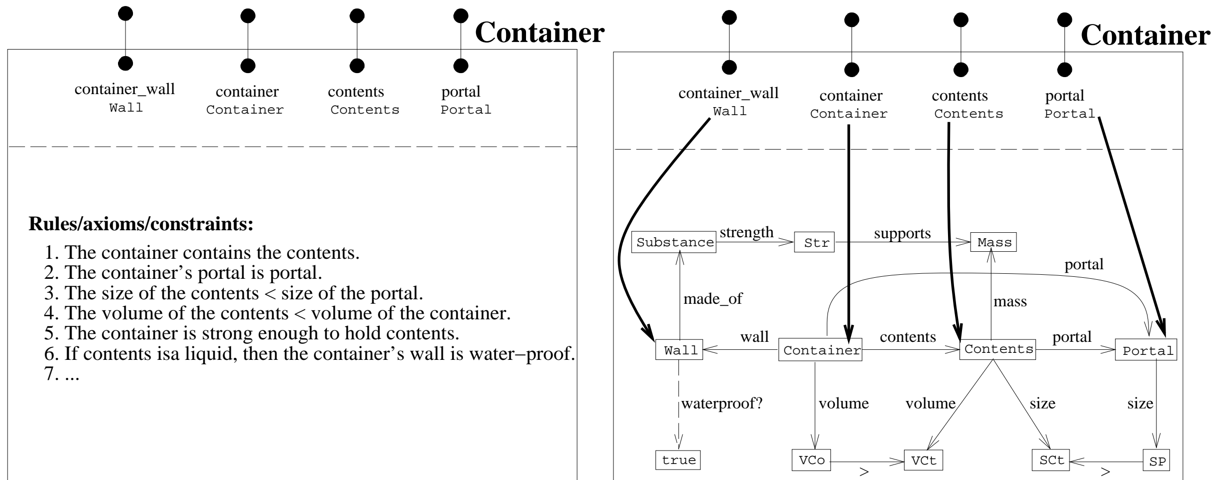


Figure 1: The `Container` component is a collection of axioms about the `container-portal-contents` system. The axioms can equivalently be viewed as a conceptual graph fragment.

---

2 Object Description Components

2.1 Components as a System of Concepts

Axioms express relationships and constraints among concepts: they are not really 'about' a single concept, but rather make statements about a set or *system of concepts*. The idea of a component, then, is to collect and package axioms about the same system of concepts. Each concept within that system is an *interface parameter* to the component. A component's parameters thus provide 'hooks' into the system of relationships the component describes.

The axioms themselves are (possibly conditional) statements of relationships among those concepts. We can view these axioms as *fragments of a conceptual graph*. An example of a `Container` component is shown in Figure 1 with its contents depicted in both axiomatic and conceptual graph forms.

In Working Note 6, we conceptually divided a component's interface into 'input parameters' and 'output blocks'. As discussed there, this division was purely conceptual: the component's role was simply to enforce a set of relationships among its terminals (parameters and blocks). This is also the case for these 'object description' components: we can provide some or all of the component's parameters, and the component will then ensure that its relationships are enforced.

2.2 Applying Components

To *connect* a component to a representation being constructed, we map concepts in that representation onto the terminals of that component. In other words, plugging in a component makes a statement that a particular set of concepts (namely those which we chose to connect to the component's terminals) exist in a particular relationship. It identifies and overlays a general system of concepts (eg. `container-portal-contents`)

on the description being constructed.

To state this in another way: introducing a component involves stating which concepts in a representation fill the *roles* of concepts in that component. For example, we might apply a `Container` component to a (representation of a) person, saying that his/her stomach fills the role of container, food fills the role of the container's contents, and his/her mouth fills the role of the portal. Note that filling a role is not the same as being 'isa' something: for example we do not assert that `food isa contents`, but instead state that `food` fills the role of `contents` in this person-as-container viewpoint.

As discussed in the previous working note, we may restrict the *structure* of data allowed at a component's terminals, and hence restrict the component's applicability. In this way, we can design special sub-classes of a component (eg. `physical-container`), which can only be applied to particular types of data.

2.3 Realms and Component Libraries

Following the GenVoca methodology, we can create families ('realms') of components which share the same standardized interface, but have different contents. Each component in a realm represents a different way that the same abstract concept can be realized. For example, we could define a realm `Container`, and then define different container components: `Container01` might describe containment of physical objects, `Container02` might describe containment of virtual objects (eg. ideas) by an agent, `Container03` might describe physical containers such as the human body (where the contents weren't put in the container though a portal), `Container04` might describe containment of liquids, etc.

The standardized interface should allow us to interchange components. However, it is unclear at this point how useful this interchangability would be in practice. As an example where it might be useful, consider a representation of `Lawnmower`, which would include a component `PetrolEngine`. We might then be interested in modifying this to represent a lawnmower which used an electic engine instead. If both `PetrolEngine` and `ElectricEngine` both used the same `Engine` interface (ie. both were in the realm `Engine`), then such plug-and-play modification would be straightforward.

2.4 Nesting Components

In representing constraints/relationships among a system of concepts, a component might itself *import* information from other components. In other words, we can connect components together to arbitrary depths.

An example of this is given in Figure 2. In this Figure, the component `Kayak` imports relationships from sub-components `Container`, `Human-Propulsion`, and `Floatation`, and combines them together to provide an overall object description. To 'import' relationships from the `Container` component (say), `Kayak` provides the mapping between its concepts and container roles (namely `hull=container`, and `seat=contents`). Note that, in this case, `portal` in `Container` has no counterpart in `Kayak`. As a result, the `portal` concept is *added* to the conceptual graph as a new node (rather than some existing node being viewed as the portal).

2.5 Components and Compositional Modelling

As a point of interest, these object description components have many similarities with *model fragments* (MFs), the basic building blocks used in compositional modelling [Levy, 1993,

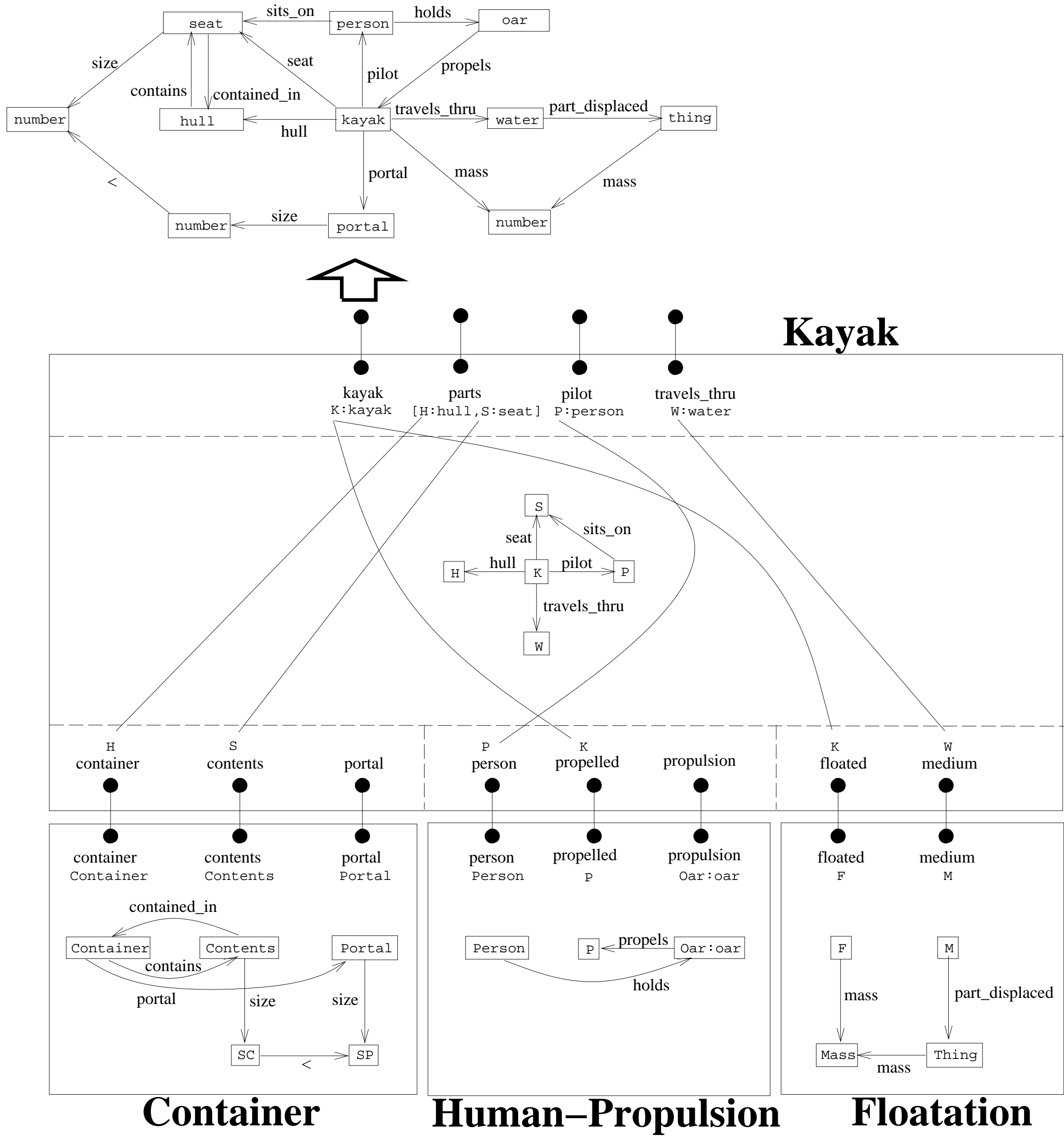


Figure 2: The `Kayak` component itself connects with three sub-components (`Container`, `Human-Propulsion`, and `Floatation`) to construct a composite object description.

Falkenhainer and Forbus, 1991]. A component's parameters correspond to a MF's *individuals* (Falkenhainer) or *variables* (Levy), and the component's axioms correspond to a MF's *relations* (Falkenheiner) or *behavior-conditions* (Levy). Structural restrictions on a component's terminals are analogous to a MF's *operating conditions* (Falkenheiner,Levy).

The GenVoca method also introduces new ideas not present in compositional modelling: in particular, the notion of standardized interfaces (realms), component interchangability, and stacking of components, has no clear analog in Compositional Modelling.

3 Discussion

We have described a new model of an object-description component (compared with the earlier component-as-an-axiom model), in which a component is a system of concepts, and components are applied by identifying how concepts fill roles in that component. This is implemented using the GenVoca model of components and composition, presented in the previous working note. We now raise several points of interest and open issues relating to this approach.

3.1 Concepts and Roles

There is a subtle but essential 'trick' this approach uses: namely to view things like `container` not as a concept but as *role* which a concept can fulfil. The reader may notice that in Figures 1 and 2, `container` never appears as a concept[1] (ie. a node in a graph), and may not even appear in the `isa` hierarchy. Instead, it is the label of a component's terminal, denoting a role which some other concept can fill. We have thus avoided (or at least reduced) the common problem of introducing a proliferation of 'role concepts' (eg. `producer`, `transportee`) and then wondering where they should reside in the isa hierarchy.

However, this raises a corresponding issue: what is a concept and what is a role? eg. is `person` a concept, or a role which some concept can fulfil? Maybe everything is a role. This is still an open issue.

3.2 Component Application as Generalized Inheritance

This approach largely replaces inheritance with component application. In fact, we can view this change as a generalization of inheritance:

Inheritance: Given a concept 'maps onto' (ie. `isa`) some general concept, we infer (inherit) new information.

Component Application: Given a *system* of concepts maps onto some other *system* of concepts (eg. `hull-seat` maps onto `container-contents`), we infer new information.

This use of *concept systems* rather than single concepts to infer new information is something which many KB systems already do (eg. the `getvals()` operation in KM,

[1] Although *variables* called `Container` exist. Note the subtle distinction: names which are lower-case denote concepts, names starting with upper-case denote variables.

subsumption-based inference in KL-ONE systems, and rule-based inference in expert systems). The difference here is that we connect concept systems by stating how concepts fill roles, rather than using `isa` relationships.

3.3 Flexible Components

In some cases, it might be inappropriate to apply every single axiom within a component. For example, if an axiom expressing a soft constraint was violated, we might still wish the composition to proceed. To do this, we would need some method for allowing axioms within a component to be ignored if conflicts were encountered. We raised this issue briefly in the earlier Working Note 3.

3.4 Which Sockets should a Component Have?

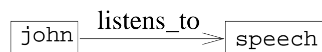
Figure 2 depicts the `Kayak` component as having three sockets (`Container`, `Human-Propulsion`, and `Floatation`). In fact, we could arbitrarily add any number of other sockets, depicting other abstractions which could be applied to `Kayak` (eg. `Transportation`, `Recreation`). It's thus not clear exactly which sockets should be explicitly built into a component: maybe we simply enumerate all the possible abstractions which could be applied, and provide a socket for each. This again is an open issue.

3.5 Reasoning with Composed Object Descriptions

By constructing object descriptions from components, many interesting forms of inference become available. In Section 3 of Working Note 6, we discussed these inference methods for composed scripts, and they are all essentially applicable to composed object descriptions also. These include: removing/including components for presentation/analysis purposes, generating text from different viewpoints, and identifying new abstractions. The reader is referred back to that working note for more detailed presentation of these.

3.6 Mapping Slots between Components

Although we can state how a concept fills a role, our method doesn't allow us to easily state how a relation (ie. slot) fills the role of another relation – in other words, we can't easily map slots together. For example, we might want to apply an `Eating` component to the graph:



We can map `john` to `eater` and `speech` to `food`, and hence conclude that the speech is now 'in' john (in some sense). The problem is that we would like to also map the relation `listens_to` to `eats`, and probably `in` to `knows`, but cannot at present without modifying the way components connect.

It may be that this is simply a technical issue related to using metaphors. However, the point at which a view becomes a metaphor is at best fuzzy. Again, this is an open issue.

3.7 How Feasible and Useful are Realms?

In the GenVoca model, by creating sets of components which share the same standardized interface (realm), we achieve component interchangability: we can simply remove a com-

ponent and replace it with another one of the same realm, and it will still connect with its parent component. While this is an intuitively nice feature, it's not completely clear how this interchangability can be exploited. In fact, it is not even clear if such standardization of interfaces can be achieved for object descriptions. For example, as mentioned earlier, we might like to create `PetrolEngine` and `ElectricEngine` components which share the same realm of `Engine`, so that we can interchange them. The problem is that the roles in a petrol engine which define the interface to the `PetrolEngine` component (eg. `fuel`, `tank`, `distributor`, `exhaust`) do not always have clear counterparts in `ElectricEngine`, and vice versa. Hence a standardized `Engine` interface may be difficult to achieve.

4 Summary

We have described briefly how the model of components discussed in the previous working note might be applied to representing object descriptions, and indicated some of the benefits and issues which result. Our next task is to investigate how this pans out in practice.

References

[Falkenhainer and Forbus, 1991] Falkenhainer, B. and Forbus, K. (1991). Compositional modelling: Finding the right model for the job. *Artificial Intelligence*, 51:95–143.

[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.

[Sowa, 1984] Sowa, J. F. (1984). *Conceptual structures : Information processing in mind and machine.* Addison Wesley.