The Trouble with What: Issues in method-independent task specifications

Enrico Motta, Zdenek Zdrahal

Knowledge Media Institute
The Open University
Walton Hall,
Milton Keynes,
MK7 6AA
United Kingdom
E.Motta, Z.Zdrahal@open.ac.uk

Abstract. In this paper we discuss some issues concerning the organization of knowledge for reuse and we critically examine the ideas of *knowledge separation* and *minimal ontological commitments*.. Because knowledge structures can play multiple roles in a domain, it is not necessarily the case that *search-control knowledge* can be neatly separated from a domain ontology. This is particularly the case when only procedural descriptions of a task are available. Because expert knowledge is often 'messy', clean separation of knowledge can be obtained only by removing the 'troublesome' structures. However, we believe that this approach does not pay off in terms of reusability. Less knowledge means less reusability. We argue that a different approach to developing reusable ontologies is needed, which does not impose 'strict' separation of knowledge, emphasizes the underlying assumptions about the available domain expertise, and does not trade knowledge for 'formal purity'. These ideas are discussed in the context of the VT elevator design problem.

1. INTRODUCTION

Over the past years the emphasis in the knowledge engineering community (and in particular in the knowledge acquisition community) has shifted from a view of expert system building as a process of mapping expert knowledge to a computational shell, to a view in which the construction of a knowledge based system (KBS) is seen as a structured development activity, characterised by a number of *models* fulfilling different roles in the application development life-cycle (Wielinga et al., 1992a; 1992b; Shadbolt et al., 1993). At the same time, libraries of reusable *domain ontologies* (Gruber, 1993a), *problem solving methods* (Mc Dermott, 1988), and *generic modelling components* (Le Roux et al., 1993; Breuker & Van de Velde, 1992) are emerging, which are transforming the development of a KBS application into a process characterised by the reuse and customization of pre-existing modelling components. These are defined at different levels of abstraction and play different roles in the application development process.

In order to maximise reuse, it is important that the modelling components which make up the emerging libraries are highly applicable, i.e. they must be generic, modular, and customizable for different domains. These properties need to be true both of the overall architecture of the KBS and the internal structure of the reusable modelling components. For instance, Gruber (1993b) proposes a number of guidelines informing the design of reusable domain ontologies and - in particular - suggests that these should "*minimize ontological commitments*". Runkel et al. (1994) discuss instead the organization of KBS and suggest that the "key to reusability" is in making explicit the separation of the knowledge about the task and the domain in hand (the *task specification*) from the "*search-control knowledge*" used by a particular problem solving method to solve the given task efficiently. Thus, problem-independent task specifications can be produced which are more reusable and to which different problem solving methods can be applied.

These two ideas - minimizing ontological commitments and knowledge separation - have informed the development of a domain ontology formally capturing the task specification of the VT elevator design problem (Gruber et al., 1994). The VT problem was originally discussed in (Marcus et al., 1988) and has been chosen in the context of the Sisyphus II initiative (Schreiber & Birmingham, 1994) as a sample problem on which to compare different approaches to the development of shareable and reusable KBS. An informal description of the VT problem is presented in (Yost, 1992).

In this paper we review some of the work carried out in the context of the Sisyphus II initiative and we discuss a number of issues concerning the organization of knowledge for reuse both in the specific case of the VT ontology and in general. In particular we wish to highlight the following two points:

- We understand the idea of minimizing ontological commitments in the specification of domain ontologies as concerning the *vocabulary* (Gruber, 1993b) of an ontology, and we agree with this approach. However, we think that this approach improves reusability only as long as it does not affect the contents of a knowledge base. If knowledge structures which should be part of a knowledge base have to be 'sacrificed' for the sake of minimizing ontological commitments - as it seems to be the case with the VT domain ontology - then the overall result is a less reusable ontology, not a more reusable one.
- The idea of knowledge separation is an important tool for achieving reusability, but it is not always the case that knowledge can be neatly separated into different categories. For instance, in some cases a knowledge structure used by a particular problem solving method - and as such to be classified as search-control knowledge - can also play a role in the task specification. This is particularly the case when a problem is described *procedurally*, by providing strategies to attain a solution, rather than *declaratively*, in terms of a complete set of criteria specifying the range of acceptable solutions. Moreover, there isn't necessarily an objective way of separating problem solving knowledge from *task knowledge* (i.e. from knowledge about the task specification). This means that, for a given domain, it is possible to produce alternative but equally valid task specifications, which exhibit different advantages and disadvantages and are based on different assumptions on the available domain expertise, for instance its degree of *completeness*. Because alternative but equally valid task specifications can be formulated, it is crucial that the assumptions underlying a particular task specification are explicitly recorded as part of its *design rationale*.

Our analysis of i) the informal description of the VT domain, ii) the formal VT domain ontology, and iii) a number of solutions to the VT problem presented at the 1994 Banff workshop provides supporting evidence for the above statements. For example the fact that different papers exhibit different views on what constitutes the VT task specification shows how difficult (and controversial) is to distinguish between task and problem solving knowledge. The problem is compounded by the fact that the different groups tackling the VT problem fail to make explicit the assumptions on the nature of the available domain expertise underlying the organization of their models. For example, in this paper we show that the VT formal domain ontology makes an 'assumption of completeness' about the existing domain expertise, which is neither present nor required by the informal description of VT and which lacks (and requires) validation. Moreover, we also show that some knowledge structures in VT play a role in the specification of both search-control and task knowledge. As a result alternative task specifications for VT can be defined which subscribe to different views on the nature of the task - in particular whether or not VT should be characterized as an optimization task - and on the completeness of the available knowledge. Finally, our analysis also shows that minimal domain ontologies are not necessarily more reusable. Because knowledge cannot always be easily separated into mutually exclusive categories, minimal ontologies can in some cases be achieved only by removing some 'messy' knowledge structures. The trouble is that this loss of knowledge might impede the use of an otherwise applicable problem solving method.

We believe that the solution to the aforementioned problems is in producing domain ontologies and task specifications which, while still obeying a declarative formulation, embed all available knowledge and make explicit the underlying assumptions (e.g. completeness) about the acquired knowledge. In particular, in contrast with the knowledge separation approach we argue that relevant problem solving knowledge should be part of a task specification, especially if experts provide only procedural task descriptions. Analogously, domain ontologies should be 'information-rich' and contain multiple, even redundant and possibly inconsistent views. In the paper we'll show how we can use these ideas to produce alternative, valid, documented, and reusable task specifications for VT.

The paper is organized as follows. In the next section we quickly summarize the 'methodological background' informing most approaches to KBS development as well as our discussion. In section 3 we critically review the current specification of the VT domain ontology and we show that it is based on strong, not validated assumptions about the available domain expertise and that it is not a 'faithful' model of the informal VT description given in (Yost, 1992). In section 4 we propose two alternative task specifications for VT which are based on different assumptions on the completeness of the available expert knowledge. In section 5 we address the optimization nature of the VT task and we outline a way of capturing it in a formal task specification. In section 6 we examine the general implications concerning the organization of knowledge for reuse which we can draw from our analysis of the VT domain. Finally, in section 7 we conclude the paper by summarizing the main points we have raised and by restating our views on reusable knowledge bases.

2. TASKS, PROBLEM SOLVING METHODS, AND DOMAIN ONTOLOGIES

The methodological framework informing most approaches to *knowledge-level modelling* (Van de Velde, 1994a) - i.e. to implementation-independent characterizations of knowledge bases - and used by us in the course of this paper characterizes a knowledge level model in terms of three loci of interest. These comprise the *task*, *method*, and *domain* perspectives. To our knowledge this view was clearly stated for the first time in (Steels, 1990) and since then - to a smaller or larger extent - most contributions to knowledge-level modelling have subscribed to it (Chandrasekaran et al., 1992; Wielinga et al., 1992a; 1992b; Puerta et al., 1994; Runkel et al., 1994).

A task perspective specifies what needs to be achieved. For instance the task in the VT elevator design problem (Yost, 1992) is to assign values to the design parameters in a way which satisfies all given constraints. VT can therefore be seen as an instance of a *parametric design problem* (Schreiber, 1994).

A method perspective specifies the knowledge used and required by a problem solving method, which provides a way of tackling a task. For instance, the document describing the VT problem (Yost, 1992) illustrates one particular approach to solving VT, which is called Propose&Revise (Marcus et al., 1988).

Finally, a domain perspective a-la-Steels provides a particular *view* over a domain. Within the knowledge sharing community the term 'domain ontology' has a slightly different emphasis, denoting reusable domain descriptions which are - as much as possible - method (and possibly task) independent. For instance the VT domain ontology contains information about parameters and components in the VT domain, as well as the associated constraints.

Thanks to this framework and to the emerging libraries of modelling components one can envisage the process of knowledge engineering as one in which an application problem is classified in terms of an existing task hierarchy, the relevant domain knowledge is encoded in terms of the primitives and ontological requirements specified by a suitable domain ontology, and, finally, a suitable problem solving method is selected from a method library, customised and optimised for the domain in hand, and applied to the given application problem. Applying a method to a domain requires a mapping process (Gennari et al., 1994) whereby the "knowledge roles in the method ontology" are instantiated by domain-level knowledge structures.

We share the view that this three-dimensional space (tasks, methods, and domains) is the correct methodological framework for knowledge engineering. However, as pointed out in the introduction, an approach based on strict separation of knowledge can lead to problems if the knowledge structures acquired from the domain experts play multiple roles in our methodological framework. This issue will be discussed in the next section in the context of the VT elevator design problem.

3. VT: A CASE STUDY IN TASK SPECIFICATION

3.1 Task and Problem solving knowledge in VT

The draft reference model produced by Guus Schreiber (1994) describes VT as a parametric design problem where the goal is to find an assignment of values to *parameters* which satisfies the set of *constraints* given in the ontolingua-based description of the VT domain (VT-Onto). Parameters in VT are organized around *components*. A separate file accompanying VT-Onto provides a sample test case in terms of an input specification, which is expressed through an assignment of values to a subset of the VT parameters. According to this view VT is the task of finding a design given information about components, parameters, constraints as well as an input specification - see figure 1.

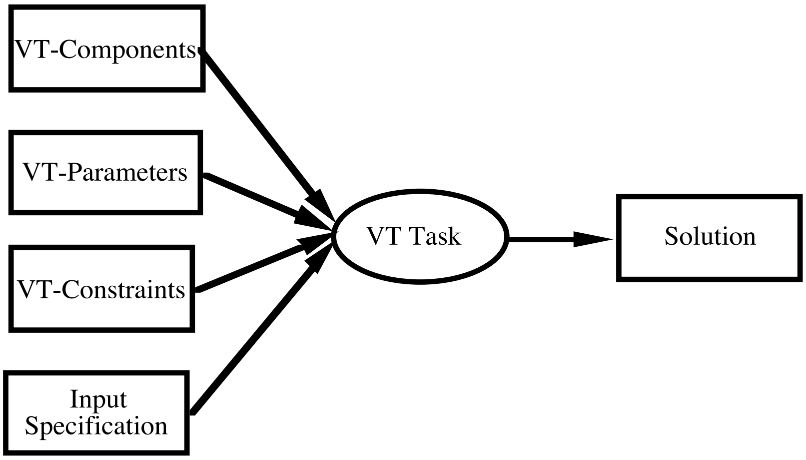

Figure 1. I/O specification of the VT task in VT-Onto

The informal description of the VT problem produced by Gregg Yost (Yost, 1992) provides instead a procedural description of the VT problem, in terms of the Propose&Revise approach used by Marcus et al. in their original VT solution - see figure 2. In particular, the Yost document specifies the following additional knowledge structures: *procedures*, to calculate new parameter values from already assigned parameters; *fixes*, which are procedures associated with constraint violations to be used when attempting to remove these; and a number of *default initial values*. These are assigned before the start of the design process and can then be modified by the application of procedures and fixes, typically (but not always, some initial values are overridden by procedures even during the 'propose' phase) to fix constraint violations.

It is clear that the task specification provided by the Yost document doesn't score very high in terms of 'separation of knowledge': task and problem solving knowledge are intertwined and the range of acceptable solutions is described procedurally rather than declaratively. VT-Onto is an attempt to solve this problem with the Yost specification. The approach taken in VT-Onto is based on the two key ideas of 'separation of knowledge' and 'minimal ontological commitments'. The result is a formal declarative task specification for VT, which is not biased towards a particular problem solving method and doesn't include any 'search-control knowledge'. To satisfy this requirement the designers of VT-Onto have not included in the domain ontology fixes and default initial values, while procedures have been transformed into declarative constraints.

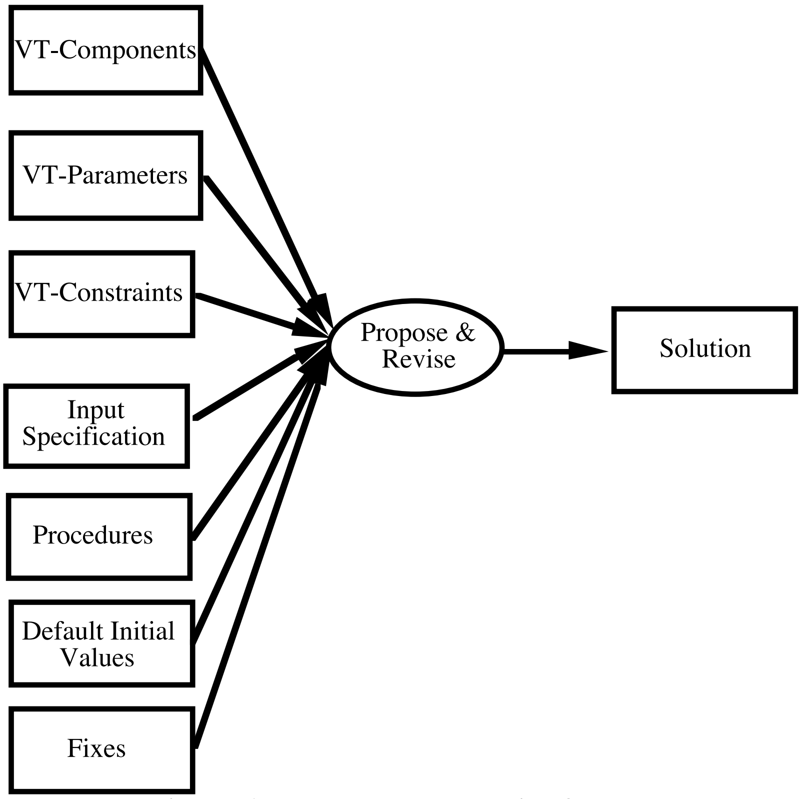

Figure 2. Propose and Revise for VT

3.2 Assessing VT-Onto

In contrast with VT-Yost (the VT problem as described in the Yost document) VT-Onto obviously provides a precise, unambiguous task specification for VT. However, this property alone does not give enough ground to conclude that VT-Onto is the 'correct' specification of the VT task. We believe that in order to achieve this goal VT-Onto also needs to fulfil the following requirements:

- It should correctly characterize the *nature* of the VT task
- It should be consistent with VT-Yost - i.e. any solution to VT-Yost should be a solution to VT-Onto and vice versa.
- It should prove to be more reusable than VT-Yost.

We believe that (to a greater or lesser extent) VT-Onto does not actually satisfy any of these three requirements.

The characterization of VT as a parametric design problem is correct, but incomplete. It does not capture the optimization nature of the VT task. As it can been seen from the description of the problem solving approach used by expert designers, it is not actually the case that any solution which satisfies the given set of constraints is acceptable. Solutions are ranked in terms of their cost and expert designers use procedures which try to approximate optimality. This essential aspect of the VT task is not present in VT-Onto.

As shown in detail in the next section, it is not the case that VT-Yost and VT-Onto define the same solution space. In particular there are solutions to VT-Yost which are not solutions to VT-

Onto and vice versa. Therefore it cannot be claimed that VT-Onto is a formal description of VT-Yost. It is rather a description of an alternative problem domain.

VT-Onto does not contain all the knowledge about the VT domain which we can obtain from VT-Yost. Moreover procedures in VT-Yost have become constraints in VT-Onto while default initial values and fixes have been removed. We believe that these aspects make VT-Onto less (rather than more) reusable than VT-Yost. In particular, turning procedures into constraints has overconstrained the theory and made the application of some fixes illegal - see section 3.3. And the fact that problem solving knowledge about VT has been removed from VT-Onto is an obstacle to the use of Propose&Revise-like approaches. As discussed in section 3.4.3 it is still possible to apply a Propose&Revise problem solver to (a revised version of) VT-Onto by performing syntactic mappings. However, it seems to us that this approach reduces instead of increasing reusability.

These points will be discussed in detail in the rest of the paper.

3.3 VT-Yost vs. VT-Onto

It is very easy to see that VT-Onto specifies a problem which is different from the one described in Yost. In particular we can easily see that there are solutions to VT-Yost which are inconsistent with VT-Onto. Amazingly, these include the test case described by the input specification associated with VT-Onto. This can be seen as follows.

The solution to the test case given in Yost specifies - among others - the following assignment of values to parameters:

counterweight_to_platform_rear = 1.75
counterweight_space = 18.25
counterweight_plate_depth = 7
counterweight_u_bracket_protrusion = 0.75

However, VT-Onto also includes a constraint, called COUNTERWEIGHT_TO_PLATFORM_REAR_C, which is violated by these values. The constraint in question is shown below.

```
(define-frame COUNTERWEIGHT_TO_PLATFORM_REAR_C
 :own-slots (
  (INSTANCE-OF constraint)
  (CONSTRAINT-ON elevator)
  (CONSTRAINT.EXPRESSION
    '(= (counterweight_to_platform_rear ?x)
      (* 0.5
       (- (counterweight_space ?x)
        (+ (counterweight_plate_depth ?x)
         (counterweight_u_bracket_protrusion ?x)))))))
 :issues ((:formerly-named cwt_to_platform_rear_calc)))
```

The problem here is that because the distinction between procedures and constraints has been lost in VT-Onto all procedures have become constraints. Therefore the set of constraints in VT-Onto is a superset of the set of constraints in VT-Yost, thus overconstraining the problem. This doesn't mean that procedures and constraints are disjoint sets. Actually the majority of procedures in VT are also constraints. However, there are exceptions to this rule and there are a number of procedures in VT-Yost which only suggest *preferred ways of computing parameter values*. They are not there to impose constraints. This point can be clarified by looking in detail at the 'meaning' of the relevant network of procedures, fixes, and constraints, which are specified in VT-Yost - see figure 3.

The procedure in VT-Yost which corresponds to the constraint counterweight_to_platform_rear_c in VT-Onto is the following:

P1: Counterweight_to_platform_rear =
0.5 (Counterweight_space – (Counterweight_plate_depth +
Counterweight_u_bracket_protrusion))

This procedure says:

P1: Place the counterweight so that the distance between the U bracket protrusion and the counterweight is the same as the distance between the counterweight and the rear of the platform.

Based on the value of Counterweight_to_platform_rear the distance from the counterweight to the rear of the hoistway is calculated as:

P2: Counterweight_to_hoistway_rear =
Counterweight_space – (Counterweight_plate_depth +
Counterweight_to_platform_rear)

VT-Yost also includes a constraint stating that the distance between the counterweight and the rear of the hoistway must be at least 0.75 inches greater than Counterweight_u_bracket_protrusion.

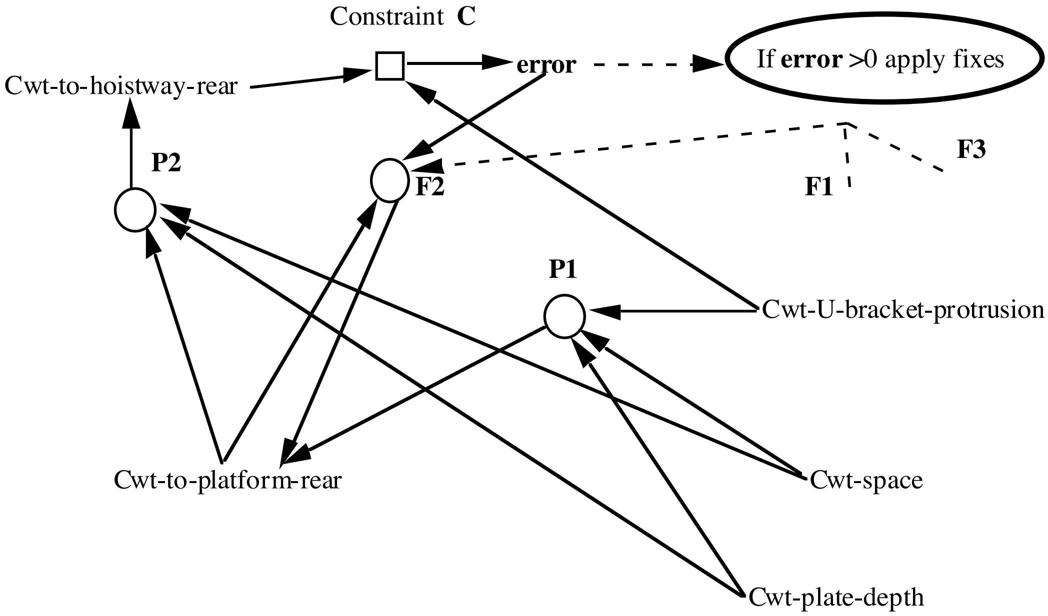

Figure 3. Dependencies in VT-Yost

If the constraint is violated VT-Yost proposes 3 fixes: F1, F2, and F3. The second one in the Yost's list says:

F2: Decrease the distance from the counterweight to the rear of the platform by the amount by which the Counterweight_to_hoistway_rear falls short of its minimum.

After substituting P1 and P2 to F2 we get a new procedure for calculating the distance from the counterweight to the rear of the platform.

P1(F2): Counterweight_to_platform_rear =
Counterweight_space – 0.75 –
(Counterweight_plate_depth + Counterweight_u_bracket_protrusion)

The design philosophy expressed in terms of these procedures is:

Preferably use strategy P1, i.e. place the counterweight to the middle between the U bracket protrusion and the rear of the platform. Alternatively, if P1 has failed - i.e. a constraint violation has occurred - use strategy P1(F2) and place the counterweight so that its distance from the rear of the hoistway is 0.75 inches.

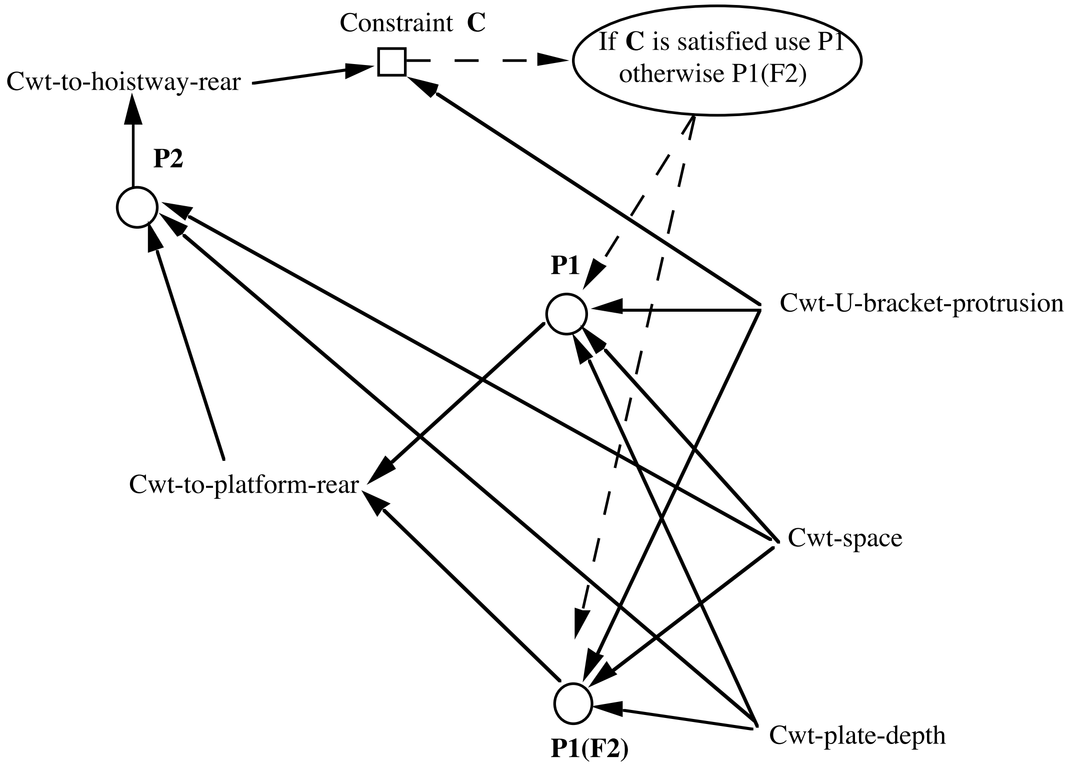

Figure 4. An alternative view of the network in figure 3.

Now, the point of all this is that P1(F2) is a procedure for calculating Counterweight_to_platform_rear, which is applied as an alternative to P1 - see figure 4. That is both P1 and P1(F2) express the same type of information, i.e. they are both strategies for calculating the distance between the counterweight and the rear of the platform. The reason why one is characterized as a procedure and the other as a fix has to do with the fact that i) experts have provided a Propose&Revise-oriented description of the VT domain; ii) procedures and fixes are the two classes of knowledge structures in the Propose&Revise framework which perform modifications to the design; and iii) P1 is preferred to P1(F2). All this has a number of important implications concerning the problem of devising a task specification for VT, which are listed below.

- Not all procedures define (hard) constraints on the solution. Because VT-Onto does not distinguish between hard and soft constraints and does not include procedures and default initial values, it follows that these ‘revisable’ procedures should not be included in VT-Onto.
- It is not necessarily the case that procedures and fixes express two epistemologically different types of knowledge. In particular, fixes are not just search control knowledge. In some cases they provide alternative, less preferred strategies for computing parameter values (i.e. alternative procedures).
- VT-Onto is not a formalization of VT-Yost. In particular the given input specification is a solution to VT-Yost but not to VT-Onto.
- VT-Yost can be seen as providing a procedural description of the space of solutions for the VT task which is based on (and therefore consistent with) the problem solving behaviour of an expert designer. Therefore, although VT-Yost only provides an implicit task specification, it nevertheless still provides one, which has the advantage of being ‘expert-

faithful'. In contrast with VT-Yost, VT-Onto provides a formal, explicit specification. The problem is that VT-Onto lacks expert validation.

In the previous section we have also argued that VT-Onto is not necessarily more reusable than VT-Yost. In the next section we'll discuss this point by reviewing some of the solutions to the VT problem proposed to the Sisyphus II workshop at Banff '94.

3.4 Different approaches to solving VT

We can classify the solutions to the VT problem presented at the Banff workshop as belonging to three classes, depending on the task specification they tackle (VT-Yost or VT-Onto) and the problem solving method they use. Of course, as we have already discussed, VT-Onto suffers from three quite serious problems: i) it is inconsistent with the test case, ii) it is not a formalization of VT-Yost, and iii) it lacks expert validation. Obviously, a posteriori, we can say that these problems affect the validity of the solutions of those groups which have tackled VT-Onto rather than VT-Yost. However these aspects are not very important here: the main issue we want to address in this section is whether VT-Onto is more reusable than VT-Yost.

3.4.1. Solving VT-Onto by means of general-purpose methods

Some solutions tackle VT-Onto using a general-purpose method. For instance approach #1 in (Runkel & Birmingham, 1994) uses a constraint satisfaction engine. This approach is correct - at least if we ignore the aforementioned problems with VT-Onto. However, even if VT-Onto were a correct model of VT-Yost, the problem of expert validation would still arise. The reason is that a general-purpose engine can of course find solutions which would never be generated by an expert. For instance, the existence of a default value, say v1, for a parameter, say P1, in VT-Yost means that any VT design should have v1 as a value for P1, unless this is impossible - i.e. unless it violates some constraint. If we tackle VT-Onto, which does not contain the information about initial default values, using general constraint satisfaction techniques, we can find solutions which do not assign v1 to P1 even when this assignment doesn't violate any constraint. The validity of such a solution relies on the assumption that the set of constraints in VT-Onto is a 'valid' - in the sense of expert-faithful - specification of the VT task. However there isn't enough evidence supporting this claim. Because VT-Yost describes the task procedurally, we don't really know whether the elicited set of constraints is large enough to cover those 'pathways' to a solution which would never be pursued by an expert. Because of the procedural nature of the acquired knowledge, it might be the case that the constraints produced by the domain experts are just those sufficient to constrain the specific set of 'pathways' which account for Propose&Revise problem solving.

An additional problem with this approach is that it does not address the optimization aspects of the VT problem - see section 5.

3.4.2. Solving VT-Yost by means of Propose&Revise

Other approaches implement their own model of the VT knowledge base from the Yost document and then tackle it using some variation of the Propose&Revise problem solving approach. For example, our own solution (Motta et al., 1994) describes a strategy called Complete-Model-then-Revise (CMR) whereby a complete, albeit inconsistent, design model is produced before starting the revision phase. The Protege-II group (Rothenfluh et al., 1994) also follow the Propose&Revise approach although they use a different strategy for fix application from the one suggested in the Yost document and used by us. The solutions in this class can be regarded as being (more or less) faithful to the expert behaviour and as tackling the (original) VT task.

3.4.3. The use of knowledge about fixes to solve VT-Onto+

Finally, a third class of solutions applies Propose&Revise to VT-Onto, for instance the CommonKADS solution (Schreiber et al., 1994), or augment a general design procedure to make use of fixes in the design of the elevator - approach #2 in (Runkel & Birmingham, 1994). Because these two approaches raise different issues we can treat them separately. Let's consider the

CommonKADS solution first, i.e. the application of a Propose&Revise problem solver to VT-Onto.

3.4.3.1 Applying Propose&Revise to VT-Onto+

The problem with this approach is that Propose&Revise cannot be applied to VT-Onto, without turning this into VT-Yost first. To clarify this point let's look at the knowledge structures used by VT-Onto and by Propose&Revise. The latter makes use of the following types of knowledge structures which are missing in VT-Onto: procedures, default initial values, and fixes. The former encodes procedures as hard constraints. We have already seen that this causes VT-Onto to be inconsistent with the test case. More importantly, in terms of reusability, it is not possible to apply F2 to VT-Onto without violating P1. That is, fixes - and therefore Propose&Revise - cannot be used with VT-Onto. A more interesting case is whether a suitably modified VT-Onto, not exhibiting these consistency problems, is amenable to be tackled by Propose&Revise. We believe the answer to this question is 'No', or at least 'not without changing the methodological approach used to design VT-Onto'. Let's see why this is the case. VT-Onto does not distinguish between hard and soft constraints. Therefore the only way to restore consistency is to remove the procedures such as P1 which are not constraints. According to our analysis VT-Onto contains eight of these 'troublesome' constraints. We call the resulting knowledge base VT-Onto+, to distinguish it from VT-Onto. Can we now apply Propose&Revise to VT-Onto+? As already said, Propose&Revise requires procedures, initial values, and fixes. If we extend VT-Onto+ to include these additional knowledge structures we basically end up with VT-Yost. But if one has to re-encode procedures, initial values, and fixes he/she does not gain that much from reusing VT-Onto+. An alternative could be that of devising a method ontology for Propose&Revise which contains the eight 'revisable' procedures which cannot be part of VT-Onto+, the given initial default values, and the VT fixes, and then defining a mapping between the set of procedures in Propose&Revise and the subset of constraints in VT-Onto+ which are also procedures. This is the approach taken both in the CommonKADS solution and in the development of an ontolingua theory which we are currently producing (Motta, in progress) to provide a formal knowledge-level model of the solution to the VT problem which we presented at the Banff '94 workshop.

```
(define-axiom reference-for-model-extension-procedures
    "This axiom specifies the role mapping between the role
     Model-extension-procedures in the P&R method specification
     and a subset of the constraints in the vt domain"
   := (= (domain-reference model-extension-procedures P&RforVT)
         (setofall ?c
                   (and (constraint ?C vt-design-problem)
                        (= (constraint.expression ?c) ?exp)
                        (or (= ?exp (listof '= @?rest))
                            (and (exists ?antecedent
                                          (= ?exp
                                             (listof
                                                '=>
                                                ?antecedent
                                                ?consequent)))
                                 (= ?consequent
                                    (listof '= @?rest))))))))
```

Figure 5. Mapping procedures in Propose&Revise to constraints in VT-Onto+

Figure 5 above shows an axiom from our ontolingua knowledge base, which specifies that the role 'Model Extension Procedures' in our P&R problem solver (applied to the VT problem) maps to the set of constraints in VT-Onto+ which exhibit a certain syntactic structure.

The problem with this approach is that it is really a sort of a kludge. Relying on the syntactic format of an expression to decide whether or not this is a procedure is not a very 'principled' approach. It can be done, but it relies on a number of implicit assumptions such as: i) the designers of VT-Onto have used a consistent syntactic style in encoding constraints, ii) there is no other constraint in VT-Yost which exhibits this structure - i.e. only procedures do. It seems to us that this approach defeats the idea of a reusable ontology providing a "content-specific agreement" (Gruber, 1993a). Moreover it seems to us an example of the sort of problems caused by the minimalist approach to designing ontologies. We think it would be much better to make explicit in VT-Onto which constraints are used by the VT designers as procedures during Propose&Revise problem solving and which are not. In this way, the ontology itself could provide a semantic rather than syntactic basis for reuse.

3.4.3.2 Augmenting a general-purpose design procedure with fixes.

The approach taken by Runkel and Birmingham in their second solution to the VT problem is very interesting because - as in solution #1 - they again use VT-Onto as their domain ontology, but this time they augment their general purpose design program with knowledge about fixes. Of course we already know that fixes cannot be used with VT-Onto without causing an inconsistency. But let's consider the case of VT-Onto+. Can we use fixes with VT-Onto+? This is a very interesting case. Of course there is no question of inconsistency here but is this a correct reuse of fixes? Again, let's look at F2. As shown in figure 4, F2 is meant to be an alternative to P1. If P1 is removed from the ontology then the only strategy which we can use to compute the distance between the counterweight and the rear of the platform is F2. This of course should work. However one could argue that this reuse of fixes goes beyond the one intended by the domain expert. F2, analogously to many other fixes, is supposed to be a less preferable alternative to calculate the value of a parameter. As such, F2 is strictly associated to P1, and we could say the latter is required if we want to give a meaning to F2. On the other hand, in the Yost description of VT, fixes are associated with constraint violations, not with procedures. Therefore, because Runkel and Birmingham only apply fixes in response to constraint violations, we should conclude that their reuse of fixes is correct with the 'intended use' by the domain experts.

The only possible criticism to this approach is that if we take the view that VT is an optimization problem, the fact that VT-Onto+ does not contain some procedures and the default initial values implies that sub-optimal solutions could be generated in cases where an expert would produce an optimal one. Moreover, the usual caveats about tackling a problem which is different from VT-Yost and about the need for expert validation apply here as well.

3.5 Summary so far

So, let's recapitulate what discussed so far. We have found that fixes do not necessarily specify only search control knowledge. This means that they cannot be a-priori excluded from the VT task specification. The current formal ontology for VT, VT-Onto, has a number of problems: i) it is inconsistent with the test case; ii) it is not a formalization of VT-Yost; and iii) assumes that the set of constraints elicited from the domain expert is sufficient to define the space of legal solutions to the VT problem. It cannot be proved whether this assumption is true or false, but it is nevertheless a pretty big assumption which should be explicitly recorded in the design rationale of VT-Onto. Moreover, we have also shown that fixes cannot be applied to VT-Onto in its present form without producing an inconsistency and that even if a consistent version of VT-Onto, VT-Onto+, is assumed, then reusability is supported only through syntactic mappings between a Propose&Revise problem solver and VT-Onto+. Moreover, VT-Onto+ is not really a very good solution. It doesn't solve the problem concerning the completeness of the set of constraints in VT-Onto, and, in terms of 'quantity of knowledge' contains even less knowledge than VT-Onto. This reduces rather than increases its reusability. We believe that a different approach to producing a task specification is needed, which is described in the next section.

4. EXPERT-FAITHFUL TASK SPECIFICATIONS

Intuitively an 'expert faithful' solution to the VT problem, as described in the Yost document, is one which a human designer would produce given a particular initial input specification.

Enforcing this requirement means - for instance - that unless constraint violations force the designer to modify his/her strategy, he/she will calculate the distance between the counterweight and the rear of the platform using the P1 procedure discussed in section 3.3.

Producing a task specification for VT which defines an ‘expert faithful’ space of solutions is difficult because this is not described by means of a set of declarative constraints, but procedurally, in terms of the ‘legal moves’ which an expert would apply when solving VT. From a formal point of view the problem with VT-Onto is that interpreting a subset of these moves as a set of declarative constraints is incorrect and leads to inconsistencies. From a knowledge acquisition point of view the problem with VT-Onto is that it assumes that the set of constraints that have been acquired from the expert, C, is *complete* and therefore any solution S(C) satisfying all constraints in C is legal, regardless of the way the solution has been produced. This assumption of completeness is a very strong one and requires expert validation.

The problem here is that in many real-world domains a legal solution space is easier to define procedurally than by giving a complete set of declarative constraints. If we take this view in the case of VT, then solutions which have not been achieved by the application of the given procedures and fixes are illegal. If we take this approach, while some solutions are eliminated extensionally (i.e. because they violate constraints), others are excluded intensionally (by not providing the means for their calculation). Let us denote with P the set of procedures which express all possible strategies for calculating parameters. P includes both the procedures which cannot be converted into constraints (e.g. P1 discussed above) and fixes (e.g. F2). Let us denote with S(P) the set of all solutions generated by P. Then the space of expert-faithful solutions to VT, Sf, is defined as $S(C) \cap S(P)$. $S(C) \cap \sim S(P)$ denotes the set of all solutions which satisfy all constraints but which cannot be reached by means of procedures from P.

An easy way to restore consistency between VT-Onto and the test case is to treat the troublesome formulas as 'soft constraints', while keeping the others as ‘hard’ ones[1]. The goal of the design process becomes then that of satisfying all hard constraints. In addition solutions which also satisfy some or all of the soft constraints should be preferred. Because default initial values play a similar role to the above formulas, i.e. they specify preferred or 'starting-point' values, they could also be treated in the same way, i.e. represented as soft constraints. We call the resulting task specification, VT-What.

VT-What doesn’t seem to suffer from the problems troubling VT-Onto. It is consistent with the test case and it is also consistent with the range of solutions which domain experts would produce. But is it? A domain expert using Propose&Revise would compute values in one of two ways. Either by means of a procedure or by means of a fix. A strict interpretation of the requirement that the range of solutions for the VT task should be expert-faithful requires that for each parameter, p_i, whose value can be computed by applying a fix $F(p_i)$, the following hard constraint C_h is applied to the task specification:

C_h: has-value (p_i, ?v) <=> ?v = Pi(p_i) Ò?v = F(p_i).

where Pi(p_i) is the procedure which computes the value of p_i and Ò denotes an exclusive or.

In this way, we can ensure that the space of solutions of the resulting task specification is expert-faithful. Because this is quite a ‘fundamentalist’ approach to providing a task specification for VT, we call this new task specification, VT-Fund.

This approach guarantees expert-faithful solutions in the sense described above. The solution spaces corresponding to VT-What and VT-Fund are shown in figure 6, which also shows how procedures and fixes of VT-Yost are converted into constraints of VT-What and VT-Fund. Solution set S1 corresponds to VT-What. Solution set S2 of VT-Fund is a subset of S1, because the set of hard constraints of VT-Fund is a superset of the set of hard constraints of VT-What and

---

[1] The very latest proposal for a revision to VT-Onto (Gennari, 1994) appears to agree with this approach.

the problem of constraint satisfaction is ‘non-monotonic’ - in the sense that increasing the number of constraints decreases the number of possible solutions. In VT-What both solutions s1 and s2 are legal, although s2 violates some soft constraint and is therefore less preferable. On the other hand solution s2 is illegal in VT-Fund.

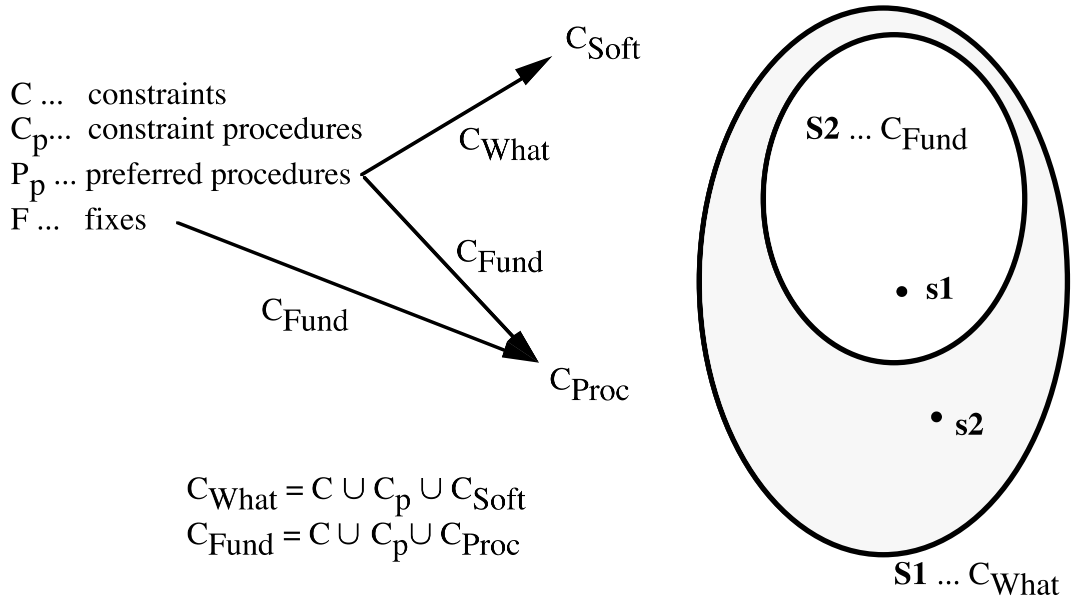

Figure 6. Solution space of VT-What and VT-Fund.

5. OPTIMIZATION ASPECTS OF DESIGN TASKS

5.1 VT-Fund*: a task specification which captures design optimization

In previous sections we have analysed various ways of defining legal solutions without paying too much attention to their ordering. However, a design task such as VT is an optimization task: we are not interested in finding just any solution; we want to find an optimal design, or at least a good enough approximation.

The optimality criterion in VT is expressed in terms of *preferences* associated with alternative strategies (procedures and fixes) and we can use these preferences as a way of computing the *design cost* of a solution. Each fix in VT is given a preference rating which ranges from 1 (most desirable) to 10 (least desirable). We can easily generalize this approach to include procedures, by saying that each procedure has rating 0 - i.e. procedures specify the most desirable design steps.

The constraint schema characterizing VT-Fund, which has been discussed in the previous section, can then be augmented to account for the numerical preferences used in VT-Yost and to provide a basis to capture the optimization nature of the VT task.

C^*_h: has-value (p_i, ?v) & preference (C^*_h) = ?c <=>
(?v = Pi(p_i) & ?c = Pref (Pi)) Ò (?v = F(p_i) & ?c = Pref (F)).

where Pi(p_i) and F(pi) are procedures which compute the value of p_i, and Ò denotes exclusive-or.

The right-hand side of the above constraint schema is a disjunction of assignment expressions. Each assignment expression is a conjunction of a value assignment and a preference. Preferences are expressed in terms of a function Pref which takes as argument a strategy for calculating the value of a parameter. As in the case of C_h, one and only one of the terms in the right hand side of the constraint has to be satisfied, for each parameter p_i. The numerical preference of C^*_h is therefore defined as the preference of the satisfied term of C^*_h.

The cost of a solution can then be expressed in terms of the cost of each instance of C^*_h, C^*_{hi}. This is in turn defined as the cost of the one and only one term in the right-hand side of C^*_{hi}, which is satisfied. The calculus for combining these individual costs will be discussed later.

Our definition of the design cost has the following consequences:

- For each complete assignment of values to parameters it is possible to decide i) whether the assignment satisfy all constraints - i.e. whether it is a solution to the design problem; and ii) what is the cost of the solution. These questions can be answered without considering the problem solving method used for designing VT.
- Since the cost is associated with strategies and not with components the design cost is not the sum of the costs of the individual design components. This is intuitively acceptable if we realise that the ultimate design goal is to satisfy certain requirements which are not associated with the cost of the individual components - e.g. the need to comply with the contract specifications.

The notion of design cost proposed here is consistent with both the preference mechanism and the Propose&Revise approach discussed in VT-Yost. Nevertheless, this definition of design cost does not depend on any particular problem solving approach. The propose steps in the Propose&Revise method correspond to the '(?v = Pi(p_i) & ?c = Pref (Pi))' term, with Pref (Pi) = 0. The revise steps are accounted for by the '(?v = F(p_i) & ?c = Pref (F))' term. Because only the revise steps have non-zero cost, the overall design cost is the cost of the design revision phase. Moreover, the design process followed by the Propose&Revise method can be represented by a *design trajectory* (Zdrahal & Motta, 1995), which consists of *assignment* (elementary propose steps) and *corrective steps* (elementary revise steps). The design cost becomes then the cost associated with such a trajectory. Many techniques have been developed for this type of optimisation, such as A*, dynamic programming, and various optimisation techniques in control theory.

In order to complete the task specification for VT-Fund* (the extension to VT-Fund which takes optimization into account) we also need to specify a calculus for computing the overall design cost from that of the individual design steps.

We won't go into too many details here and we will just point out that the way costs are combined in the VT design problem is non-Archimedean in nature. By this we mean that a non-Archimedian combining function (Zdrahal & Motta, 1995) is needed here in order to formalize the intuitive notion that n design steps of cost j are still cheaper than a single step of cost k, with k > j, no matter how big n. The interested reader can find more on this topic in (Zdrahal & Motta, 1995).

The task specification for VT-Fund* is now complete. In the next section we discuss a number of additional issues concerning the 'how' of a KBS which tries to tackle VT-Fund. That is we say something about search-control knowledge in design optimization tasks.

5.2 Global vs. local optimality

Minimizing the design cost is a problem of large scale optimisation. In general it should be solved in the multidimensional space defined by the design parameters since all parameter values may affect the design cost. Moreover the cost-based function may be of an arbitrary shape, i.e. it may have traps in the form of local minima. Ideally we would like to find a global minimum for the function but, unfortunately, it is very difficult to find a computationally efficient method (Zdrahal & Motta, 1995). For a human designer such problem is simply unsolvable. For this reason when developing techniques to tackle VT-Fund* we need to follow a 'myopic' approach in which we try to find solutions which are only locally optimal. This is not a problem because the task specification of VT-Fund* specifies a global optimality criterion in terms of preferences associated to individual steps. The bottom line is that the solution composed of locally optimal steps will be sufficiently suboptimal (and expert-faithful).

6. DISCUSSION

The actual details of VT-Onto and VT-Yost are probably of interest only to a restricted set of researchers who are working on the Sisyphus-II initiative. However, we believe that the discussion in this paper has highlighted a number of issues affecting the definition and reuse of task specifications, which go beyond the specific case of VT and are of general applicability.

6.1 On the separation of knowledge

If we look at the various solutions to the VT problem presented at the '94 Banff workshop, we can see that every single group regarded fixes as knowledge about a specific problem solving method, i.e. part of the method dimension. Possibly only the paper by Poeck et al. (1994) refers to them as "part of the domain knowledge". However, as shown in this paper, there are fixes which are more than just search-control knowledge. In particular fixes such as the one discussed in section 3.3 play a role analogous to procedures, i.e. they provide a strategy for improving the design. This kind of fixes should be part of the domain ontology. Moreover, if one takes a 'fundamentalist' view on the meaning of the procedural knowledge acquired from the VT domain, then only solutions whose design steps are consistent with the ones defined by the expert are legal (VT-Fund). Of course alternative approaches can also be taken and we have discussed a modification to VT-Onto, called VT-What, which does not require the design steps of legal solutions to be strictly faithful to those used by the domain experts. The important aspect of VT-What is that it is based on a strong assumption of completeness which ideally should be validated.

Another aspect concerning the separation of knowledge is that deciding the role that a knowledge structure plays in a system, whether it is part of the task specification or search-control knowledge, also depends on assessing the nature of the task. For example, in this paper we have argued that VT is really an optimization problem. If one takes this view, then it follows that i) the design cost is associated with the design strategies, ii) the function to combine the costs of individual design steps is non-Archimedean in nature, and iii) fixes need to be part of the task specification.

The bottom line here is that although the idea of knowledge separation is an important tool to achieve reusability, it must be kept in mind when producing reusable KBs that the knowledge acquired from experts doesn't necessarily fall easily into our modelling categories and experts do not give us nice declarative specifications. They intertwine tasks and method specifications together in messy procedural descriptions. Of course our job as knowledge engineers is to acquire and reconstruct this knowledge according to our methodological frameworks, turn it into consistent theories, and represent it in a way to maximise reuse. But this process is to a large extent judgmental and alternative task specifications can be produced, which are based on different views on the nature of the task and the available domain expertise.

6.2 On minimal ontologies

VT-Fund can be solved by any applicable PSM but, as discussed in section 3.4.3, we don't see the point of having to perform clumsy syntactic mappings in order to apply Propose&Revise to a declarative specification of the VT ontology. Both the semantics and the pragmatics of knowledge structures should be explicitly represented in a domain ontology so that users do not have to infer them from syntax. It seems to us that this problem stems from an 'over-zealous' interpretation of the "minimal ontological commitment" guideline (Gruber, 1993b). We believe that this should apply to the vocabulary of an ontology not the contents. It seems to us that rich ontologies containing multiple, even redundant, and possibly inconsistent views of a KB would be more reusable than minimalistic, knowledge-deprived ones. For instance, in the case of the VT ontology, we don't see why the constraints which are defined as procedures in VT-Yost should not be explicitly tagged.

6.3 On the design rationale of a KB.

We have extensively discussed in this paper the fact that VT-Onto defines a space of solutions which is different from VT-Yost and that this choice implies making a strong assumption on the completeness of the set of constraints acquired from the domain experts. We have also argued that if such an approach is taken, then its underlying assumptions should be explicitly recorded in the design rationale of the task specification. A recent paper by Walter Van de Velde (1994) is relevant to this discussion because it emphasizes the need to move from a view of libraries as repositories of models to one which sees libraries as repositories of “modelling episodes”. This view has many implications. An obvious one is that all the assumptions and the argumentation process used in constructing a model should be explicitly recorded in the model. For instance all assumptions about the nature and coverage of the available expert knowledge should be explicitly recorded. To this purpose we have defined a language which is specifically designed to enable knowledge engineers to record the design rationale during the development of knowledge-level models (Stutt & Motta, 1995).

7. CONCLUSIONS

In this paper we have critically examined some issues concerning the design of reusable knowledge bases in the context of the VT elevator design problem. We have highlighted some problems with the current version of the VT formal specification, in particular emphasizing the following points: i) it does not reflect the optimization nature of the VT task; ii) it makes some strong assumptions about the completeness of the acquired domain expertise; iii) it does not contain all the domain knowledge which plays a role in the VT task specification; and iv) it doesn’t correctly model the existing informal description of the VT domain. These problems make it less reusable. In this paper we have addressed these problems and we have shown how alternative task specifications could be produced which comprise all relevant domain knowledge, model the optimization aspects of the VT problem, and are consistent with the procedural description of the VT design task. In particular we have shown that alternative assumptions on the nature of the task and the available domain expertise can lead to alternative, but equally valid task specifications. We believe that this example shows the need to follow a more flexible approach to producing task specifications, which does not impose ‘strict’ separation of knowledge, emphasizes the underlying assumptions about the available domain expertise, does not trade knowledge for ‘formal purity’, and possibly ensures that domain ontologies reflect alternative views. More knowledge means more reusability.

ACKNOWLEDGEMENTS

This paper would not have been possible without the efforts of all those involved in the Sisyphus I and II initiatives. In particular we would like to acknowledge Gregg Yost and Jay Runkel with whom we have had extensive discussions about the VT domain which have helped us to clarify our ideas. We would also like to thank John Domingue for providing comments on an earlier draft of the paper.

REFERENCES

Breuker, J. A. and van de Velde, W. (1993). The CommonKADS Library. ESPRIT Project P5248 Deliverable M2, University of Amsterdam and Free University of Bruxelles.

Chandrasekaran, B., Johnson, T.R. and Smith, J.W. (1992). Task-Structure Analysis for Knowledge Modelling. Communications of the ACM, 35(9), Sept. 1992. 124-137.

Gennari, J. H., Tu, S. W., Rothenfluh, T. E., Musen, M. A. (1994). Mapping Domains to Methods in Support of Reuse. Proceedings of the 8th Banff Knowledge Acquisition Workshop, Banff, Canada, 1994.

Gennari, J. H. (1994). Personal communication. 1994.

Gruber, T. R. (1993a). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, *5(2)*, 1993.

Gruber, T. R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. (1993b). Technical Report KSL 93-04, Stanford: Knowledge Systems Lab., Computer Science Dept. Stanford University, 1993.

Gruber, T. R., Runkel, J. T., and Olsen, G. R. (1994). VT Domain Ontology. Available by FTP from ksl.stanford.edu/pub/knowledge-sharing/ontologies/. Stanford: Knowledge Systems Lab., Computer Science Dept. Stanford University, 1994.

Le Roux B., O'Hara K., Outtandy S., Shadbolt N., Laublet P. and Motta E. (1993). The VITAL Library of Knowledge Modelling. *VITAL document DD215*. Project VITAL (ESPRIT II Project P5365). Onera.

Marcus S., Stout J. and McDermott J. (1988). VT: an expert elevator designer that uses knowledge-based backtracking. AI Magazine, 9(1), 1988. 95-112.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In Automating Knowledge Acquisition for Expert Systems, Marcus (ed), Kluwer Academic.

Motta, E., O'Hara, K., Shadbolt, N., Stutt, A. and Zdrahal, Z. (1994). A VITAL Solution to the Sisyphus II Elevator Design Problem. Proceedings of the 8th Banff KA for KBS Workshop, Vol 3, 1994.

Poeck, K., Fensel, D., Landes, D., and Angele, J. (1994). Combining KARL and Configurable Role-limiting Methods for Configuring Elevator Systems. Proceedings of the 8th Banff Knowledge Acquisition Workshop, Banff, Canada, 1994.

Puerta, A. R., Egar, J. W., Tu, S. W., and Musen, M. A. (1992). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. Knowledge Acquisition 4(2), 1992, 171-196.

Rothenfluh, T. E., Gennari, J. H., Erikson, H., Puerta, A. R., Tu, S. W., and Musen, M. A. (1994). Reusable Ontologies, Knowledge-Acquisition Tools, and Performance Systems: PROTEGE-II Solutions to Sisyphus-II. Proceedings of the 8th Banff Knowledge Acquisition Workshop, Banff, Canada, 1994.

Runkel, J. T. and Birmingham, W. B. (1994). Solving VT by Reuse. Proceedings of the 8th Banff Knowledge Acquisition Workshop, Banff, Canada, 1994.

Runkel, J. T., Birmingham, W. B., Balkany, A. (1994). Separation of Knowledge: a Key to Reusability. Proceedings of the 8th Banff Knowledge Acquisition Workshop, Banff, Canada, 1994.

Schreiber, G. (1994). Reference Model for the VT Domain. Unpublished draft memo, 1994.

Schreiber, G. and Birmingham, W. B. (1994). Sisyphus II - VT Elevator design Problem. Volume III of the Proceedings of the 8th Banff Knowledge Acquisition Workshop, Banff, Canada, 1994.

Schreiber, G., Terpstra, P., Magni, P., and van Velzen, M. (1994). Analysing and Implementing VT using COMMON-KADS. Proceedings of the 8th Banff Knowledge Acquisition Workshop, Banff, Canada, 1994.

Shadbolt, N., Motta, E., and Rouge, A. (1993). Constructing Knowledge Based Systems. *IEEE Software*, November 1993.

Steels, L. Components of Expertise. (1990). AI Magazine. 11(2), 1990. 29-49.

Stutt, A. and Motta, E. (1995). Recording the design decisions of a knowledge engineering community to facilitate re-use of design models. Proceedings of the 1995 Knowledge Acquisition Workshop, Banff, Canada, 1995.

Van de Velde, W. (1994a). Issues in Knowledge Level Modelling. Proceedings of the 8th Banff KA for KBS Workshop, Vol 2, 1994a. 38-1 - 38-11.

Van de Velde, W. (1994b). A Constructivist View on Knowledge Engineering. Proc. 11th European Conf. on AI. Ed. Cohn, A. London: Wiley and Sons, 1994b. 727-731.

Wielinga B. J., Schreiber A.Th. and Breuker J. (1992a). KADS: A Modelling Approach to Knowledge Engineering. In *Knowledge Acquisition* 4(1) pp.5-53, 1992.

Wielinga, B., Van de Velde, W., Schreiber, G. and Akkermans, H. (1992b). The CommonKADS Framework for Knowledge Modelling. Proceedings of the 7th Banff KA for KBS Workshop, Vol 2, 1992.

Yost, G. R. (1992). Configuring Elevator Systems. DEC Project Report. 1992.

Zdrahal, Z and Motta, E. (1995). An In-depth Study of Propose & Revise Problem Solving. Proceedings of the 1995 Knowledge Acquisition Workshop, Banff, Canada, 1995.