

# REUSE: REVISITING SISYPHUS-VT

Derek Sleeman, Trevor Runcie & Peter Gray

Computing Science Department  
The University of Aberdeen  
Aberdeen AB24 3UE  
Scotland UK

email:{sleeman truncie pgray} @csd.abdn.ac.uk

**Abstract.** Reuse has long been a major goal of the Knowledge Engineering community. The focus of this paper is the reuse of domain knowledge acquired for an initial problem solver, with a further problem solver. For our analysis we chose a knowledge base system written in CLIPS based on the propose-and-revise (PnR) problem solver, and which had a lift/elevator knowledge base (KB). Given the nature of the problem solver, the KB contained 4 components, namely an ontology, procedural statements which specify how the artifact, the lift, could be enhanced/modified, a set of constraints to be satisfied, and a set of fixes to be applied when constraint violations occurred. These 4 components were first extracted manually, and were used with both an Excel spreadsheet and a constraint problem solver (ECLiPSe) to solve a range of tasks. The next phase was to implement ExtrAKTor which extracts the same 4 knowledge sources virtually automatically from the CLIPS knowledge base (held by Protégé), and transforms these so that they are usable with a number of problem solvers. To date Excel & ECLiPSe have been selected, and again we have demonstrated that the resulting systems are able to solve a variety of lift configuration tasks. This is in contrast to earlier work which produced abstract formulations of the problem but which were unable to perform reuse of actual knowledge bases.

## 1 Introduction

Reuse has long been a major goal of the Knowledge Engineering community. The vision being that if knowledge sources/bases about particular topics, and domain-independent problem solvers were available, then it should be possible to create a new Knowledge base System by selecting from these components and in some way linking them. This vision was partly materialised in the early days of Expert Systems, when a number of domain independent “Inference Engines” were implemented and used with a wide range of domains. For instance, the EMYCIN [1] shell was used with Knowledge Bases (KBs) for Infectious Diseases and Civil Engineering - to mention just 2 applications.

Subsequently, the Expert System community in the later 70’s / early 80’s described a range of Problem Solving methods which, they claimed, would cover the whole

spectrum of problem solving, [2, 3]. The important theme of articulating and defining PSMs was developed subsequently, for example, by the KADS project [4, 5] (section 2); however, as we shall see, these activities produced a largely theoretical framework. Reuse was still seen as an issue in 2000 when the Advanced Knowledge Technologies (AKT) project included it as one of the challenges it intends to address in supporting the KB lifecycle, [6]. The other 5 challenges being: Knowledge Acquisition/Capture, Knowledge Modeling, Knowledge Retrieval, Knowledge Publishing and Knowledge Maintenance.

On the other hand, Reuse has become a reality for the software engineer, who regularly, once the specification of a system has been finalised, starts the implementation process by searching for suitable (Java) libraries. The vision of the Knowledge Engineer is that, having built at considerable cost (largely due to the cost associated with building the domain Knowledge Base) a KBS which is able to *design*, say a lift, it is highly desirable to reuse most/all of the domain knowledge, when developing a KBS to *diagnose* faults in the same domain. Further, the vision has also always been that this process should be relatively straightforward and could be handled by a domain expert rather than by a highly specialised Knowledge Engineer/Researcher in Knowledge Representation. The following paragraphs give a scenario for a user-friendly Reuse system.

A technician has access to a domain ontology & related design information (such as procedural upgrades, constraints & fixes) necessary to complete a design task. The technician invokes a Problem Solving (PS) agent to provide a plausible configuration. The PS agent then selects an appropriate Problem Solver. Subsequently, the Problem Solver (or the agent) recognizes that certain initialization information is missing and prompts the technician for it. The PS Agent then ideally provides a design solution; or prompts the user for further information; or reports that there is no solution. The important aspects of this scenario are that the technician should:

- not require a high level of computing science expertise
- not require detailed experience of the Problem Solvers

### **1.1 Opportunities and Challenges for Knowledge Engineering in the context of the Semantic Web**

The opportunities which the world-wide-web provides for Knowledge Engineering is very considerable in principle as the types of components involved, once made available on the web, can then be reused by a sizeable number of users. Additionally, a further mode of operation is possible where domain-independent inference engines are made available as web services; then to use such a service it is only necessary to “send” the web service, Knowledge Source(s) in the required format and an answer would be returned. The great challenge of the Web, on the other hand, is being able to locate *appropriate* domain-independent problem solvers and domain-dependent Knowledge Sources. This requires, for instance, that the capabilities of the problem solvers are adequately described; various languages such as DAML-OIL & OWL-S, have been developed to allow services to be described, and these could be used again

in this context. Such approaches require the developer of the service to describe its purpose, as well as the nature of the inputs required by the service and the nature of the output produced. Searching for an appropriate Knowledge Source is a related problem. However, some sophisticated search engines such as Swoogle [7, 8] and OntoSearch [9, 10] enable users to specify keywords to describe the sorts of content required of Knowledge Sources such as Ontologies; OntoSearch additionally allows users to specify other desirable properties which should hold in the retrieved Knowledge Sources such as structure between classes, properties of some key attributes, etc.

The structure of the rest of the paper is as follows: Section 2 describes the VT (Vertical Transport) design task, the Sisyphus-II challenge [11], outlines related work, and gives an overview of constraint satisfaction techniques; Section 3 reports the work done to manually extract domain knowledge from a CLIPS [12] Propose and Revise algorithm for the lift domain, and describes a system developed to (semi)-automatically extract the same domain knowledge; and Section 4 reviews the work and suggests possible future work.

## 2 The VT problem, Sisyphus-II Challenge, and Related Work

### 2.1 VT Problem

The Vertical Transportation (VT) domain is a complex configuration task involving the interaction of all of the components required to design a lift (elevator) system. These components are shown in Figure 1.

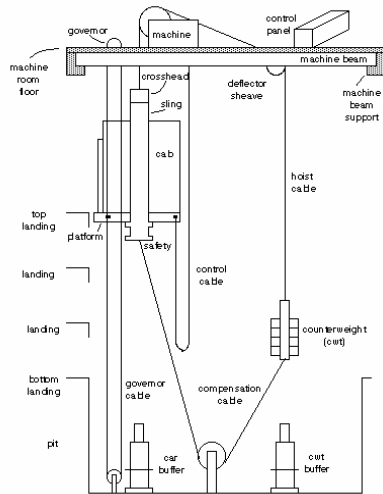


Fig. 1. VT System Components.

The parameters such as physical dimensions, weight and choice of components are regulated by physical constraints. The VT domain [13] was initially used to solve real-world lift design by Westinghouse Elevator Company. The original VT Domain included some potentially conflicting fixes. For example, if the “traction ratio” is exceeded then one fix is to *increase* the “car-supplement-weight”, but if the “machine-groove-pressure” is exceeded, one fix is to *decrease* the “car-supplement-weight”. The conflict occurs if both the “traction ratio” and the “machine-groove-pressure” are exceeded. Should the “car-supplement-weight” be increased or decreased? This original VT domain knowledge was simplified and the above conflicts removed to form the Knowledge Acquisition Sisyphus-II Challenge. The Sisyphus [11, 14, 15] version of the VT domain was created so that researchers would have a common KB for experimentation. It is the Protégé version of the VT system from Stanford University which has been used in this project, [16].

## 2.2 A Review of Problem-solving Methods (PSMs)

Problem-solving methods (PSMs) describe the principal reasoning processes of knowledge based systems (KBs). Benjamins and Fensel [17] provides a summary of PSM related research up to 1998; examples of PSMs are “heuristic classification” [18] and “propose-and-revise” [19]. In the mid-80s, researchers realised that PSMs all have goals to be achieved, actions needed to achieve the goals, and knowledge required to perform the actions. For the next decade or so there was a detailed investigation of the nature and number of PSMs, before the field attempted to analyze in detail the subcomponents which make up the PSMs.

It was appreciated that the benefits that accrue from such a PSM library could be significant, as new KBSs constructed using proven reusable components rather than building from scratch should reduce development time and improve reliability. This area of research has been most notably investigated through the KADS/CommonKADS Expertise Modeling Library, the Protégé PSM Library, and also the IBROW project. For details of these approaches see [20].

The biggest issue with these three approaches is that none support the execution of a KBS. Having stated that CommonKADS has hundreds of PSMs [2], the same Fensel and Motta paper states “None of these methods is implemented”. A recent critique of CommonKADS [3] suggests that in addition to the three levels of knowledge description a fourth type of “operationalisation” or “code” should have been included since there is no executable PSM code. There is a similar issue with the Protégé-2000 PSM Library, as the PSM librarian webpage [21] states “the current version of the PSM Librarian tab does not support actual activation”. Fensel and Motta [2] highlights a major issue with work in the PSM and PSM library field as “Configuring the optimal problem solver may have an order of magnitude higher complexity than the problem that should be solved by the problem solver.” In our view Problem Solving Methods serve little purpose if they do not support the development of operational KBS [22].

## 2.3 An Overview of Constraint Satisfaction Techniques

Constraint Satisfaction [23] techniques attempt to find solutions to constrained combinatorial problems, and there are a number of efficient toolkits in a variety of programming languages. The definition of a constraint satisfaction problem (CSP) is:

- a set of variables  $X = \{ X_1, \dots, X_n \}$ ,
- for each variable  $X_i$ , a finite set  $D_i$  of possible values (its domain), and
- a set of constraints  $C_{\langle j \rangle} \subseteq D_{j_1} \times D_{j_2} \times \dots \times D_{j_t}$ , restricting the values that subsets of the variables can take simultaneously.

A solution to a CSP is a set of assignments to each of the variables in such a way that all constraints are satisfied. The main CSP solution technique is consistency enforcement, in which infeasible values are removed from the problem by reasoning about the constraints. ECLiPSe [24] is a software system for the cost-effective development and execution of constraint programming applications.

## 3 Extraction of Components & their Reuse

### 3.1 Introduction & Manual Reuse

The initial task undertaken here was to analyse the VT-Sisyphus code which consisted of 20,000 lines of CLIPS code that included domain knowledge and a version of the Propose and Revise (PnR) algorithm. This was done manually in order to reveal the underlying processes involved, which could then be automated. The manually extracted data was then reused in both an Excel “emulator” spreadsheet and ECLiPSe constraint solver.

### 3.2 Semi-Automatic Reuse

The next stage of the project was to create a tool that could semi-automate the extraction and transformation processes that had been demonstrated manually. The tool developed is known as ExtrAKTor and was designed to support the idealized design session defined in Section 1.

The starting point for the process was the VT domain ontology represented in Protégé. This ontology was part of the Stanford solution to the VT-Sisyphus challenge [11]. The development tools used to implement ExtrAKTor were the Protégé PrologTab and GNU Prolog. PrologTab provides a tight coupling between the Protégé environment and GNU Prolog. A Prolog environment was selected because of the availability of PrologTab and its syntactical similarities to ECLiPSe. Conceptually there are three phases to the basic ExtrAKTor process, namely extraction of knowledge, creation of a new KBS, and KBS execution.

#### 3.2.1 Extraction of Knowledge

Firstly the various knowledge components have to be extracted from the original KB. They are essentially: the ontology, the procedures to enhance/modify the artifact

(in this case a lift), the domain constraints and the fixes. ExtrAKTor assumes that the constraints and fixes have been gathered together in a single class. This was the case for the Stanford Protégé based VT ontology. To initiate the analysis ExtrAKTor requires the analyst to provide slots names for the constraints and a parent class for components information. In VT, for constraints these slots are “constraint.condition”, “constraint.variable”, “constraint.expression”; for components the class is “elvis-models”. Because different ontologies use different names for these entities it is at present necessary for the analyst to identify these variables. Alternatively, if these names were to be standardized then this stage could be done automatically. Clearly, the current situation where the analyst is required to have some knowledge of the implementation (the domain variables) is at odds with the vision outlined in Section 1. Further, it has been assumed that variable names in the formulae correspond to slot names in the ontology. PrologTab allows these slots to be accessed directly using “mapped” predicates. More details can be found in Runcie [22].

### 3.2.2 Creation of a New KBS (CSP)

The next phase in the process was the creation of a new KB from the extracted knowledge for use with an appropriate domain-independent problem solver, in this case ECLiPSe. The processing of the component information is simply a case of taking as input the component data from the intermediate ASCII file which is formatted as a Prolog list structure and reformatting and outputting this in a Prolog database format. The constraint information requires a little more work since the expressions are stored in a CLIPS format these need to be parsed and reformulated in a Prolog format (including variable names). Once all of this information has been stored to a file, the KB is ready for the execution phase.

### 3.2.3 Execution of the New KBS (CSP)

In order to execute the ECLiPSe KB, a goal must be specified. The goal, entered as a variable name or a list of variable names, can be used to find the value for a single variable, or many variables. So the “ALL” option solves for all (unknown) variables; to solve for variables *Counterweight\_to\_platform\_rear*, and *Counterweight\_space*, the user must enter the goal *vt(Counterweight\_to\_platform\_rear, Counterweight\_space)*.

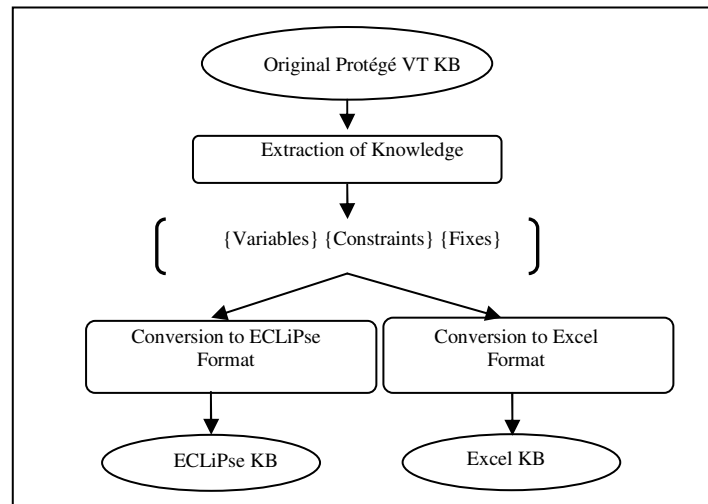
One of the main advantages of ECLiPSe for use with this class of design problem is that it supports real numbers. Additionally, if a number cannot be precisely determined, a range will be reported. e.g. “2.345...3.456”. This is a major benefit because it gives the user feedback as to the “flexibility” of the resultant designs and the user can then further refine the input. After refinement, the user can treat this as input for a new scenario (i.e. a sub-design) and can re-execute the solver again.

Similarly the information which has been extracted from the original CLIPS program could also be used with other problem solvers. The processes involved in producing KBs are captured in Figure 2.

### 3.2.4 An Extension to the ExtrAKTor System

In the simple process the desired goal is specified after the entire knowledge base has been extracted and the new KBS created. If the goal is specified earlier in the process, it is possible to generate a simplified and more focused KBS only containing

the knowledge required to solve the desired problem. The size of the original KB is probably the deciding factor in determining the most efficient choice. For a large ontology or KB, creating a simplified and focused KBS is probably more appropriate. The disadvantage of this approach is that every goal specification requires a separate extraction process, and a separate KBS is produced for each.



**Fig. 2.** Overview of ExtrAKTor and the stages needed to create both ECLiPse and Excel KBs.

## 4 Discussion & Future Work

The main focus of this paper is the extraction of components from the VT-Sisyphus domain so that they can be reformulated and re-used in conjunction with different problem solvers. There have already been some early successes in this project, as the automated extraction of an ontology, formulae, constraints, and fixes has demonstrated. Significantly, this extracted information has been used to solve real problems with different problem solvers. Further, as far as we have ascertained, no other research has used constraint solvers with the VT domain.

As noted earlier the KBS based on the ECLiPse system executes the standard lift configuration task very rapidly (0.01 second). However, it is important to note that the equational nature of this problem may make it highly suited for solution by ECLiPse and hence delivers the exceptional execution speed observed. Additionally, the VT problem may be under-constrained, or even deterministic, and this could also explain the exceptional performance.

Planned future work includes:

- Introducing fixes into the CSP formulation of a task
- Dealing with mutually inconsistent fixes; these fixes were removed in the Sisyphus-II simplification of the VT domain KB
- Using ExtrAKTOR with additional KBs (it is planned to use the UHAUL task)
- Developing a Java based open source version of ExtrAKTOR
- Making ExtrAKTOR a web-service

## 5 Acknowledgements

This project has benefited from an association with the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC) which is sponsored by the UK's Engineering and Physical Sciences Research Council (EPSRC) under grant GR/N15764/01. We acknowledge useful discussions of Research issues with Tomas Nordlander & David Corsar. Additionally, we also acknowledge discussions with various members of the Protégé team at Stanford University, including Mark Musen who also made available their version of the Sisyphus-VT code.

## 6 References

1. Bennett, J. and Englemore, R., *Experience using EMYCIN*, in *Rule-Based Expert Systems*, B. Buchanan and E. Shortliffe, Editors. 1984, Addison-Wesley. p. 314-328.
2. Fensel, D. and Motta, E. *Structured Development of Problem Solving Methods*. in *11th Knowledge Acquisition for Knowledge-Based Systems Workshop*. 1998. Banff, Canada.
3. Valente, A., Breuker, J. and Van de Velde, W., *The CommonKADS library in perspective*. IJHCS, 1998. **49**: p. 391-416.
4. Schreiber, G., et al. *CML: the CommonKADS conceptual modeling language*. in *EKA94*. 1994.
5. Breuker, J. and Van de Velde, W., *The CommonKADS Library for Expertise Modeling*. 1994, Amsterdam: IOS Press.
6. AKT: <http://www.aktors.org/publications/reuse/>, 01-August-2004.
7. Li Ding, et al. *Swoogle: A Search and Metadata Engine for the Semantic Web*. in *Thirteenth ACM Conference on Information and Knowledge Management*. 2004. Washington DC.
8. Swoogle: <http://swoogle.umbc.edu/>, 01-March-2006.
9. Thomas, E., et al. *OntoSearch: a Semantic Web Service to Support the Reuse of Ontologies*. in *AI-2004*. 2004. Cambridge.
10. OntoSearch: <http://www.ontosearch.org/>, 01-March-2006.
11. Yost, G., *Sisyphus 1993 - Configuring Elevator Systems*. 1994, SMI.
12. CLIPS: <http://www.ghg.net/clips/CLIPS.html>, 02-Dec-2005.
13. Marcus, S., Stout, J. and McDermott, J., *VT: An Expert Designer That Uses Knowledge-Based Backtracking*, in *AI Magazine*. 1988. p. pp. 95-111.

14. Schreiber, A.T. and Birmingham, W.P., *The "Sisyphus" knowledge-acquisition benchmark experiments*. International Journal of Human-Computer Studies, 1996. **44**(3/4): p. 275-280.
15. Sisyphus II: <http://ksi.cpsc.ucalgary.ca/KAW/Sisyphus/>, 02-December-2005.
16. Protege VT Sisyphus Ontology: <ftp://ftp-smi.stanford.edu/pub/protege/S2-WFW.ZIP>, 21-April-2006.
17. Benjamins, R. and Fensel, D., *Editorial: problem-solving methods*. IJHCS, 1998. **49**: p. 305-313.
18. Clancey, W.J., *Heuristic Classification*. Artificial Intelligence, 1995. **27**: p. 289-350.
19. McDermott, J., *Preliminary Steps Toward a Taxonomy of Problem-Solving Methods*. Automatic Knowledge for Acquisition for Expert Systems, 1998: p. 225-254.
20. Runcie, T., Sleeman, D. and Gray, P.M.D., *Pragmatic Approaches to Knowledge ReUse: the Sisyphus-VT Case Study*. 2006, University of Aberdeen: Aberdeen.
21. PSMTab: <http://protege.stanford.edu/plugins/psmtab/PSMTab.html>, 02-Dec-2005.
22. Runcie, T., *PhD Thesis (in preparation)*, in *Department of Computing Science*. 2006, University of Aberdeen: Aberdeen.
23. Smith, B.M., *A Tutorial on Constraint Programming*. 1995, University of Leeds School of Computer Research.
24. ECLiPSe: <http://eclipse.crosscoreop.com/eclipse/>, 03-April-2006.