

# KBS Development through Ontology Mapping and Ontology Driven Acquisition

**David Corsar**

Department of Computing Science  
University of Aberdeen  
Aberdeen, UK  
dcorsar@csd.abdn.ac.uk

**Derek Sleeman**

Department of Computing Science  
University of Aberdeen  
Aberdeen, UK  
dsleeman@csd.abdn.ac.uk

## ABSTRACT

The benefits of reuse have long been recognized in the knowledge engineering community where the dream of creating knowledge based systems (KBSs) on-the-fly from libraries of reusable components is still to be fully realised. In this paper we present a two stage methodology for creating KBSs: first reusing domain knowledge by mapping it, where appropriate, to the requirements of a generic problem solver; and secondly using this mapped knowledge and the requirements of the problem solver to “drive” the acquisition of the additional knowledge it needs. For example, suppose we have available a KBS which is composed of a propose-and-revise problem solver linked with an appropriate knowledge base/ontology from the elevator domain. Then to create a diagnostic KBS in the same domain, we require to map relevant information from the elevator knowledge base/ontology, such as component information, to a diagnostic problem solver, and then to extend it with diagnostic information such as malfunctions, symptoms and repairs for each component. We have developed MAKTab, a Protégé plug-in which supports both these steps and results in a composite KBS which is executable.

## Categories and Subject Descriptors

I.2.1 [Artificial Intelligence]: Applications and Expert Systems; I.2.6 [Artificial Intelligence]: Knowledge Acquisition

## General Terms

Algorithms, Design, Human Factors

## Keywords

Reuse, KBS, Problem Solvers, Ontology, Mapping, Knowledge Acquisition

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

K-CAP'07, October 28-31, 2007, Whistler, British Columbia, Canada.  
Copyright 2007 ACM 978-1-59593-643-1/07/0010...\$5.00.

## 1. INTRODUCTION

A Knowledge Based System (KBS) is an AI system which imitates human problem solving by applying some form of reasoning to domain knowledge stored in a knowledge base. During the 1980s the KBS community recognised several problem solving methods (PSMs) which they believed could meet the reasoning requirements of the majority of KBSs [2]. This perspective initiated a new approach to KBS development in which KBSs were built by selecting and configuring various components from repositories. Typically such repositories would be libraries of domain knowledge, libraries of problem solvers (PSs), and so on. When a new KBS is required the developer would select the appropriate components from these libraries and, preferably with very little effort, configure them to work together to solve his problem. Ideally an automated agent or broker would perform much of the process for the developer, reducing development times and costs. However, despite considerable research aimed at achieving this dream, it is, for many reasons, still to be fully realised. While various projects strived to meet this goal, each produced their own approach to the problem, developed different, often incompatible, technologies to support their approach; moreover, none of them have fully executable implementations.

We believe that the primary reasons why previous approaches were not completely successful was due to a) the lack of a standard formalism for domain knowledge, b) the lack of a standard to represent rules, and c) the lack of tools which allow standardised rule sets to operate over standardised representational schema.

Technologies and standards have progressed however, and OWL<sup>1</sup> now provides the standard ontology language; additionally SWRL<sup>2</sup> provides a standard which can be used for defining rule based PSs; and tools such as Protégé<sup>3</sup> provide a mature, extendable framework for both creating and using ontologies. In fact, Protégé provides a good environment for building a KBS by combining reusable components, as it includes extensive import facilities for differ-

<sup>1</sup><http://www.w3.org/2004/OWL/>

<sup>2</sup><http://www.w3.org/Submission/SWRL/>

<sup>3</sup><http://protege.stanford.edu>

ent ontology languages along with several reasoning plugins (called tabs) which allow various types of reasoning to be performed against an (instantiated) ontology. One of the most mature reasoning tabs is JessTab<sup>4</sup> which allows Jess<sup>5</sup> production rules to be executed against a knowledge base (instantiated ontology). By using these standards and technologies, we are able to focus on developing tools for achieving the rapid re-configuration of a domain ontology, typically developed for one type of PS, for use with another type of PS; additionally we do not need to implement a reasoning engine.

Our methodology for achieving KBS development through reuse consists of two phases. After selecting a generic PS and a domain ontology (possibly initially part of an existing KBS where the ontology has been decoupled), the user maps any relevant domain knowledge from the domain ontology into a form usable by the generic PS. The tool's knowledge acquisition (KA) process then uses the requirements of the generic PS and the mapped domain knowledge to guide the acquisition of the domain specific problem solving knowledge. After completing the KA process, a KBS is formed from the selected PS, the acquired domain specific rules, and the appropriate domain ontology. Further, if the domain specific rules contain new concepts then the KA system uses these to enhance the corresponding domain ontology.

The mapping stage is a common technique for enabling reuse of existing domain knowledge from a domain ontology. However, the focused KA is unique to our approach and is, we believe, critical to successful reuse, as a domain ontology developed for one type of PS will not normally contain all the information required by another. For example, one would *not* expect a rule set nor domain ontology designed for a configuration KBS to contain all the information required by a diagnostic KBS.

This paper is structured as follows: first we discuss previous approaches to KBS development which use domain ontologies/PS configurations; then we outline our approach and the tool we have built; we then discuss some further implementation details; provide an evaluation of our tool/methodology; and conclude with a summary and future plans.

## 2. RELATED WORK

There have been various projects which have looked at automatically configuring PSs and domain ontologies: three of which are PSM Librarian, CommonKADS, and IBROW3. The EXPECT project, which deals with extending KBSs, is also relevant. Additionally, Neches *et al.* [8] provide a good overview of the challenges faced with this type of KBS development, along with possible solutions. For a more detailed discussion of these projects, see forthcoming thesis [4].

<sup>4</sup><http://www.ida.liu.se/~her/JessTab/>

<sup>5</sup><http://www.jessrules.com>

## 2.1 CommonKADS

CommonKADS [2, 14] is the result of a major European project, which focused on developing a complete KBS development methodology, encompassing project management, organisation analysis, and knowledge and software engineering.

The CommonKADS methodology specifies a process by which a KBS is developed through the construction of a *product model*, which describes the state of an organisation after the planned KBS has been put in place. The product model is composed of six separate sub-models, one of which, the *expertise model* describes the reasoning component of the KBS. [14, chap. 6] provides outlines for 11 different Problem Solving Methods (PSMs), including assessment, diagnosis, and design. Each outline (template knowledge model) is composed of: a general description of the method, an abstract specification of the reasoning algorithm, a suggested domain schema, and sample variations of the method. When building the reasoning component, the developer selects the relevant template knowledge model, which provides him with guidance for implementing the PS, along with the outline of an example domain KB. The developer then has to implement the PS, define and populate the domain KB and “assemble” the PS and domain model into a working system.

## 2.2 PSM Librarian

PSM Librarian [5] provides a KBS development methodology based on reuse and configuration of domain ontologies and problem solving knowledge. The methodology is based on four types of ontology: domain, method (PSM), PSM description, and mapping; and involves the user selecting a domain ontology and a method ontology and providing a set of mappings between the two by instantiating the mapping ontology.

The domain ontology is a PSM-independent description of a particular domain, possibly taken from some library. The method ontology provides a signature for the PSM, describing the roles and requirements the domain knowledge must fulfil. Again, ideally the method ontology will be taken from a PSM library, which is described, accessed and queried through the PSM ontology. The UPML (Unified Problem-Solving Method Development Language) meta-ontology [10] is used to describe the available PSM libraries. The mapping ontology [11], a mediating layer in the architecture, provides a bridge between the domain and method ontologies. Once all mappings have been defined (manually) they can be executed by the mapping executioner sub-system of the PSM Librarian, with the resulting instantiated method ontology providing the KB for the PS to reason over. There is currently no support for executing the configured KBS however.

## 2.3 IBROW3 Project

The main objective of the IBROW3<sup>6</sup> project was the development of an architecture that facilitated an “intelligent bro-

<sup>6</sup><http://hcs.science.uva.nl/projects/IBROW3/home.html>

kering service” to produce a KBS by reuse of “third-party knowledge-components through the WWW.” UPML was developed to support the definition of knowledge-components such as domain ontologies, PSMs, PSM libraries and tasks (problem specifications). The process involved the user providing the intelligent broker with the description of a task and domain ontology, the broker would then select a suitable generic PSM, configure it to work with the user’s ontology, execute the new KBS, and return the solution to the user. Due to the challenges of doing all these steps automatically, the project did not fully achieve its aim; however UPML has been used by other approaches (including PSM Librarian) and has contributed to the IRS<sup>7</sup> (Internet Reasoning Service) project.

## 2.4 EXPECT

The EXPECT system [1] supports a user when extending and customising an existing KBS to fit their requirements. Typically this involves taking a KBS which works in some domain, and customising it to work in another by adding new rules and methods relevant to the new domain. Users are guided by a series of KA Scripts [15], which ensure all necessary information is acquired from the user.

## 2.5 Shortcomings

Although earlier approaches have made significant theoretical contributions, their implementations were inadequate as they lack suitable tools and in some circumstances require the users to perform complex mapping and/or system configuring tasks manually. The CommonKADS approach requires the developer to build multiple models of the organisation (up to six different models are typically required), each of which can take a considerable time to develop and require considerable documentation, which can add substantial overheads to the KBS development project [6]. Further, due to a lack of good quality support tools, the CommonKADS methodology provides the developer with only minimal support with the difficult task of developing these models and assembling them into a complete system.

The PSM Librarian approach also has some shortcomings: it requires the user to provide many mappings with little support; it does not provide the PS with knowledge from sources other than the domain ontology; and currently does not appear to provide/create an executable KBS. The IBROW3 project attempted to perform each step in the development process completely automatically by having a broker select a suitable domain ontology and PS and then configure the two to work together; an ambitious task which we believe is still unachievable. The EXPECT system provides an environment for adding and extending methods to an existing KBS for use in another domain. This requires the user to have a very good understanding of the original KBS, the methods it contains and the consequences of adding/changing methods, which typically requires the developer to be very familiar with the original KBS implementation.

<sup>7</sup><http://kmi.open.ac.uk/projects/irs/>

## 3. OUR APPROACH

We have developed a practical methodology for building KBSs through reuse. Our methodology performs automatically as much as possible, while supporting the user when he/she needs to make decisions. We have built MAKTab, a plug-in for the Protégé environment which implements our methodology. MAKTab uses ontology mapping techniques to suggest possible mappings between the domain ontology and the chosen generic PS; and includes a guided KA component which uses the requirements of the generic PS and the knowledge acquired from the mapping phase to aid the user extend the generic PS to their chosen domain.

This approach builds on our previous work on reusing rule sets with multiple ontologies [3]. In that project, we developed PJMappingTab, a plug-in for Protégé which helps the user in configuring a JessTab rule set designed for one ontology for use with another. JessTab rules must name specific concepts from the ontology they use: a requirement which ties them to that particular ontology. PJMappingTab uses lexical similarity metrics to suggest mappings between the concepts referenced in a rule set and those in a new ontology. After the user accepts the mappings, the original rule set is updated to reference the concepts in the new ontology; the resulting system can then be executed.

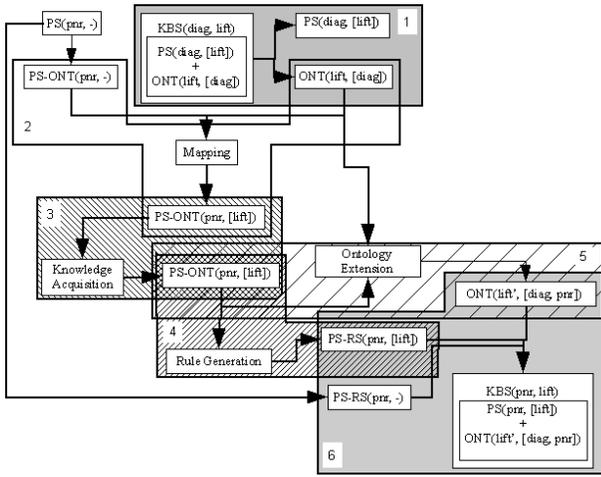
### 3.1 Illustrative Example

Throughout this paper, we use the tasks of developing KBSs dealing with elevator configuration and elevator diagnosis to illustrate our approach. Elevator configuration has been used as a KBS task by various projects. Marcus *et al.* [7] developed the original system, SALT, and others, such as the Sisyphus-2 KA Challenge [13] have used it as a way of evaluating KA tools and approaches. Both of these projects used a propose-and-revise PS combined with knowledge of elevator components to produce design specifications of complete elevator systems which meets a set of requirements such as building dimensions, minimum capacity and lift speed. The propose-and-revise method uses knowledge of components, their properties, values these properties can have, constraints on these values, and fixes for violated constraints to produce, if one exists, an acceptable combination of components. In outline its algorithm is:

1. Propose a design, if no proposal returned then exit with failure,
2. Verify proposal, if OK then exit with success,
3. If unsuccessful, systematically attempt to repair all the constraint violations with the sets of fixes provided.

To perform this successfully, the algorithm requires three types of domain specific knowledge/rules, which are used in its execution:

1. **Configuration rules** which specify how a list of sub-components can be combined to form a complete system.



**Figure 1: Outline architecture and algorithm for reusing the  $ONT(lift, [diag])$  from  $KBS(diag, lift)$  with  $PS(pnr, -)$  to produce  $KBS(pnr, lift)$ , see section 3.1 for more details.**

2. **Constraints** which specify restrictions between the various components of the configuration.
3. **Sets of Fixes** which should be applied to remedy particular violated constraints.

So from this perspective, a KBS is composed of domain knowledge and problem solving knowledge. For example, the elevator configuration KBS described above, henceforth referred to as  $KBS(pnr, lift)$  (see Table 1 for our notation), is composed of two components: an elevator domain ontology designed for propose-and-revise,  $ONT(lift, [pnr])$ ; and a propose-and-revise PS,  $PS(pnr, [lift])$ . The latter is defined as a rule set which captures the generic propose-and-revise algorithm ( $PS-RS(pnr)$ ), an ontology to capture the essential components of the propose-and-revise algorithm (i.e. the constraints, the fixes, etc.) namely  $PS-ONT(pnr, -)$ , and domain specific rules,  $PS-RS(pnr, [lift])$ .

Elevator diagnosis can also be a complex task, which involves linking observed symptoms to component malfunctions. Again, in our formalism such a KBS,  $KBS(diag, lift)$ , contains two components: a diagnostic lift ontology,  $ONT(lift, [diag])$  specifying components, component malfunctions, symptoms and possible causes; and the diagnostic PS,  $PS(diag, [lift])$ .

We have acquired a working version of both the  $KBS(pnr, lift)$  and  $KBS(diag, lift)$ . Both systems were acquired as CLIPS<sup>8</sup> KBSs, and we have re-engineered them to work within the Protégé/JessTab environment. Both KBSs were acquired from independent sources; and we have been very careful not to alter their domain and PS ontologies to avoid being accused of designing them to work just within our framework.

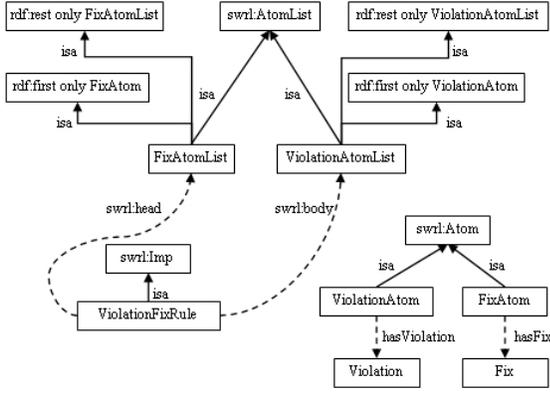
<sup>8</sup><http://www.ghg.net/clips/CLIPS.html>

Abbreviation	Meaning
PS	Problem Solver (PS-RS + PS-ONT)
PS-RS	Rule Set which implements a PS
PS-ONT	Ontology used by a PS
ONT	Domain Ontology
KBS	Knowledge Base System (PS + ONT)
pnr	Propose-and-Revise
diag	Diagnosis
lift	Lift domain
$PS(pnr, -)$	Domain independent pnr PS, which is composed of $PS-ONT(pnr, -)$ and $PS-RS(pnr)$
$PS(pnr, [lift])$	pnr PS developed in the context of the lift domain, composed of components: $PS-ONT(pnr, -)$ , $PS-RS(pnr)$ , and $PS-RS(pnr, [lift])$
$PS-RS(pnr)$	Rule Set which implements the generic pnr algorithm
$PS-RS(pnr, [domain])$	Rule Set which implements the domain specific pnr rules for the domain <i>domain</i> , e.g. $PS-RS(pnr, [lift])$ is the set of lift specific pnr rules
$PS-ONT(pnr, -)$	$PS-ONT$ which defines the concepts used by $PS-RS(pnr)$ and $PS-RS(pnr, [domain])$
$PS-ONT(pnr, [lift])$	$PS-ONT$ which defines the concepts used by $PS-RS(pnr)$ and $PS-RS(pnr, [lift])$ instantiated with relevant lift knowledge (components and/or rules)
$ONT(lift)$	Lift domain ontology
$ONT(lift, [pnr])$	Lift domain ontology used by $PS(pnr)$
$ONT(lift', [pnr, diag])$	Lift domain ontology used by $PS(pnr)$ and extended with knowledge for $PS(diag)$
$KBS(pnr, lift)$	A KBS using the pnr PSM for the lift domain. $KBS(pnr, lift)$ is composed of 2 linked components: $ONT(lift, [pnr])$ and $PS(pnr, [lift])$

**Table 1: Definition of the notation used to describe KBSs in our work.**

Our methodology is such that the user should be able to extract the domain ontology from an existing KBS and rapidly configure a further generic PS to work with it to produce a new KBS. Figure 1 illustrates one such example in which a diagnostic lift ontology,  $ONT(lift, [diag])$  (extracted from the composite KBS) and generic propose-and-revise (configuration) PS,  $PS(pnr, -)$  are configured to work together, producing a new configuration KBS in the lift domain,  $KBS(pnr, lift)$ . Our algorithm for achieving this is to:

1. Split  $KBS(diag, lift)$  into  $ONT(lift, [diag])$  and  $PS(diag, [lift])$  (this is easy in the Protégé/JessTab implementations).
2. Map relevant domain knowledge in  $ONT(lift, [diag])$  to  $PS-ONT(pnr, -)$  (extracted from  $PS(pnr, -)$ ), to produce an initial  $PS-ONT(pnr, [lift])$  (see section 3.3).
3. Use  $PS-ONT(pnr, [lift])$  with the KA tool to acquire propose-and-revise rules for the lift domain, to produce an extended  $PS-ONT(pnr, [lift])$  (see section 3.4).
4. Generate  $PS-RS(pnr, [lift])$  from  $PS-ONT(pnr, [lift])$  (see section 3.4.2).



**Figure 2: Part of the PS ontology for propose-and-revise, showing the ViolationFixRule.**

5. If any new domain concepts are introduced in step 3, add these to  $ONT(\text{lift}, [\text{diag}])$  to create  $ONT(\text{lift}', [\text{diag}, \text{pnr}])$  (see section 3.4.2).
6. Combine  $PS(\text{pnr}, [\text{lift}])$  (which is composed of  $PS\text{-}RS(\text{pnr}, [\text{lift}])$  and  $PS\text{-}RS(\text{pnr})$ ) with  $ONT(\text{lift}', [\text{diag}, \text{pnr}])$  to create  $KBS(\text{pnr}, \text{lift})$  (see section 3.4.2).

### 3.2 Describing PS Requirements

As described above, every generic PS is composed of two components: the rule set which provides the implementation of the generic PS rules and the ontology which defines the rule components and the structure of the rule set. To be useful, a generic PS must be supplied with domain knowledge, which the final KBS can use in the reasoning process. We have developed a simple PS ontology which can be used by developers to describe PSs. The purpose of this ontology is to describe the types of domain knowledge it requires and the domain rules that (when acquired) relate the generic PS to a particular domain.

Our basic PS ontology extends SWRL, adding a top-level class for domain knowledge and three classes which MAKTab uses as part of its focused KA phase. Various SWRL subclasses are used to describe the structure of domain rules which the generic PS requires. For example, the  $PS\text{-}ONT(\text{pnr}, -)$  contains a  $ViolationFixRule$  class, which states that if a constraint has been violated, then a particular fix should be applied. Figure 2 shows the ontology components which define this rule.  $ViolationFixRule$ , a  $swrl:Imp$  subclass, has its  $swrl:body$  (rule antecedents) constrained to be a  $ViolationAtomList$ , which only contains  $ViolationAtoms$  which describe a constraint violation and its  $swrl:head$  (rule consequents) constrained to be a  $FixAtomList$ , which only contains  $FixAtoms$ , each of which describe a possible fix for a violated constraint(s). These restrictions ensure any  $ViolationFixRules$  comply with the “IF *violation* THEN *fix*” nature of the rule. Table 2 provides an example of how this is used to define a rule.

Instance Type	Instance Name	Property Description
<b>Constraint</b>	<i>c1</i>	$\text{exp}(\text{cab-weight-total} < \text{motor-supported-weight})$
<b>Violation</b>	<i>v</i>	$\text{hasConstraint}(c1)$
<b>Violation-Atom</b>	<i>va</i>	$\text{hasViolation}(v)$
<b>Violation-AtomList</b>	<i>val</i>	$\text{rdf:first}(va) \text{ rdf:rest}(\text{Nil})$
<b>Fix</b>	<i>f</i>	$\text{hasAction}(\text{upgrade-motor})$
<b>FixAtom</b>	<i>fa</i>	$\text{hasFix}(f)$
<b>FixAtomList</b>	<i>fal</i>	$\text{rdf:first}(fa) \text{ rdf:rest}(\text{Nil})$
<b>Violation-FixRule</b>	<i>vfr</i>	$\text{swrl:body}(val) \text{ swrl:head}(fal)$

**Table 2: An instantiation of the ontology shown in Figure 2 to specify the rule that if constraint *c1* is violated then apply the fix which upgrades the motor.**

As mentioned previously, the PS ontology has three classes which the developer can use to provide MAKTab with extra information for guiding the process of acquiring new domain rules. The main class,  $ProblemSolver$ , serves several purposes. Firstly, it provides the developer with a place to provide a textual description of the PS and how users can customise it to their domain (important if the PS is to be used by non-knowledge engineers); secondly it allows the PS developer to provide MAKTab with some information about the PS, such as which rules it should start the KA process with, and an implementation of any generic PS rules and functions. For example, in propose-and-revise these include rules for checking if the goal state has been reached, or if no further configurations are possible. It also tells MAKTab which Java class it should use to convert the acquired rules into an executable format, which allows the developer to include any generic PS code along with that for the acquired rules<sup>9</sup>. The ontology also features a  $RuleMetaClass$  class which can be used for describing interdependencies between rules. For example, in  $PS(\text{pnr}, -)$  it is typical to expect each rule describing a constraint to have at least one associated rule describing a fix that should be applied if that constraint is violated. A  $PSConcept$  class which is the superclass of all concepts used by the PS (for example  $Doors$ ,  $Motor$ , and  $Cables$  in the lift domain), is also specified.

We argue that this basic ontology provides a suitable structure with which generic PSs can be defined; and that these can later be configured to work with any domain. A further justification for the design of this ontology, which allows one to develop additional PSs, is given in [4].

<sup>9</sup>We provide converters into JessTab format, but this feature allows future developers to define converters for other formats.

### 3.3 Ontology Mapping

Mapping is the first step in acquiring domain knowledge for the generic PS. It provides the user with the facility to reuse any existing domain (ontology) knowledge already available, in the development of their new KBS. This is achieved by mapping the knowledge contained in the user's domain ontology to the PS's ontology (for example, PS-ONT(pnr, -) in the case of PS(pnr, -)). We expect the main knowledge acquired from the mapping stage to relate to domain entities, which are represented by the `PSConcept` class (and its subclasses) in the PS ontology, which are then used in the development of domain rules in the KA stage. The main challenge for the user in the mapping stage is determining which concepts in their ontology map to concepts in the PS ontology, and how these mappings are defined. As such, we have designed the mapping tool in MAKTab to have a simple interface, and to be extendable so that we can incorporate new mapping requirements in the future as needed. In the remainder of this section, we discuss the mapping tool with respect to the four criteria defined by Park *et al.* [11] for describing ontology mapping tools (these points are discussed in sections 3.3.1 to 3.3.4).

#### 3.3.1 Mapping Power/Complexity

This refers to the expressive power and complexity of the mappings supported by the tool. As the number and type of transformations (mappings) supported is the limiting factor in this type of knowledge reuse, our tool supports an extendable range of mapping types. These include simple transformations (the renaming of a property); the concatenation of multiple properties (from a class in the domain ontology) into a single target (PS class) property; and more powerful mappings such as copying a class and class-to-individual mappings. In the later an individual of a PS class is created to represent a class (and its associated individuals) of the domain ontology. This mapping type allows the user to, for example, specify that all doors (as represented by individuals of the `Doors` class in `ONT(lift, [pnr])`) have the same symptoms, malfunctions and repairs and should therefore be represented as one individual of the `PSConcept` class in `PS(diag, [lift])`. We believe we currently provide a suitable collection of mapping types to meet the requirements of users; the tool has been designed however so that new mapping types can be easily incorporated as needed.

#### 3.3.2 Mapping Scope

The scope of a mapping defines the range of domain classes it can be applied to. In order to reduce the number of mappings the user is required to define, the user can specify if the mapping should be applied only to the class it is defined for, or if it can be recursively applied to that class's subclasses, with the option of specifying how deep it should be applied.

#### 3.3.3 Mapping Dynamicity

Dynamicity refers to when and how the mappings are invoked. In MAKTab mappings are invoked when the user is satisfied with the defined mappings, and instructs the tool to apply them.

#### 3.3.4 Mapping Cardinality

The cardinality of an ontology mapping tool specifies the nature of the mappings it supports. MAKTab supports N:1 mappings, allowing multiple domain classes to be mapped to a single problem solver class. This is necessary to allow, for example, many subclasses of the `Component` domain class (such as `Door`, `Motor`, etc.) in `ONT(lift, [pnr])` to be mapped to the single `PS(diag, -) Component` class. N:N mappings could be supported if required in the future.

#### 3.3.5 Automatic Suggestions

MAKTab aims to reduce the number of mappings the user is required to provide. Allowing inheritance of mappings can help; as can automatically suggesting property renaming mappings to the user. MAKTab suggests mappings by attempting to match class and property names in the domain ontology with those in the PS. These suggestions are produced by three types of equivalence tests: firstly finding identical names and those pairings with a similarity value, as determined by the string similarity metrics library `Simmetrics`<sup>10</sup>, over a user set value; then matching those with a (user set) percentage of common constituents; and finally `WordNet`<sup>11</sup> suggests appropriate synonyms. This algorithm is based on that of `PJMappingTab` [3]. We recognise that ontology mapping/matching is an active research field<sup>12</sup> and have designed the suggestion component to be extendable.

Once the user thinks that he has defined all the necessary mappings for the ontology, MAKTab applies the mappings to the ontology, converting the instance data into the form required by the generic PS. The composite KBS is then usually executed with several typical tests. At any stage the user is free to return and define/apply more mappings if necessary.

### 3.4 Focused Knowledge Acquisition

Having completed the mapping stage, a focused knowledge acquisition process is then used to extend the knowledge available to the PS. This process uses the requirements of the PS, specified by its PS ontology, along with the knowledge gained about the domain from the mapping stage to guide the acquisition of the additional rules it requires to function in the chosen domain. Currently the KA process interacts with a human user who is *assumed* to be capable of providing the required information.

#### 3.4.1 Acquiring Rules

The KA tool of MAKTab uses the information provided in the PS ontology, described above, to guide the acquisition of the domain specific rules<sup>13</sup>. This acquisition is based

<sup>10</sup><http://www.dcs.shef.ac.uk/~sam/simmetrics.html>

<sup>11</sup><http://wordnet.princeton.edu/>

<sup>12</sup>See <http://www.ontologymatching.org> for details on ontology mapping research.

<sup>13</sup>If the PS developer has not provided information such as which rules to start KA with or rule interdependencies, MAKTab attempts to work out interdependencies by examining the restrictions on the rules' `swrl:body` and `swrl:head` properties it assumes conse-

```

AssignValueRule-1
IF doors-opening-type has no value
AND doors-model-name = "COSS"
THEN doors-opening-type IS "centre"

AssignValueRule-2
cab-weight-total =
doors-weight + cab-casing-weight +
safety-features-weight

ConstraintRule-1
IF total-cab-weight > motor-supported-weight
THEN assert a violation of this constraint

```

**Figure 3: Example AssignValueRules and ConstraintRule for the PS(pnr,-) in the lift domain.**

on the concepts that have been gained from the mapping stage (which can easily be added to by the user at any stage during KA, if required, by creating new individuals of the PSConcept class or relevant subclass). The KA tool presents the user with the list of PSConcept individuals that have been acquired, allowing the user to select one of them and then start building the rules relevant to it. In the case of PS(pnr, -), the first rule to be acquired is an AssignValueRule, which describes how to calculate a value for a component of the elevator or a variable in the configuration. Figure 3 shows two example AssignValueRules: first a simple rule (AssignValueRule-1) for setting the value of the doors-opening-type and secondly, a more complex one (AssignValueRule-2) for assigning a value based on values of other parameters. AssignValueRules are related to ConstraintRules which define the constraints on a value, and assert a violation if one has occurred, such as ConstraintRule-1 in Figure 3. ConstraintRules in turn are related to ViolationFixRules, which we discuss in section 3.2. MAKTab takes the user through defining each rule in turn, providing suggestions for antecedents/consequents where possible. Typically this involves defining the antecedent of the first rule based on the selected (PSConcept) individual, then using the consequent of one rule as the antecedent of a further related rule (if the relevant type restrictions allow). A sample protocol for acquiring the rule defined in Table 2 is provided in Figure 4.

### 3.4.2 Generating the KBS

After all the necessary rules have been acquired from the user, MAKTab can then generate an executable KBS. It does this by producing the domain specific PS rules, for example PS-RS(pnr, [lift]), and converts the `swrl:Imp` (sub-)classes' individuals into executable code, by using a Java class provided by the PS developer. MAKTab presents the user with the results of the conversion, for the user to copy-and-paste into the relevant inference engine (for example, to the JessTab text entry window). For completeness we also allow the user to extend their original domain ontology

quents derive new facts, and any rules which use the same type of fact (atom type) in their antecedent are thought to be related.

```

SYS: Creating a new ViolationFixRule. Current antecedents:
Violation hasConstraint (exp (cab-weight-total <
motor-supported-weight)). Add another antecedent?
USER: No
SYS: Currently, there are no consequents. Add one?
USER: Yes
SYS: Creating a new FixAtom. What is the action for this Fix-
Atom?
USER: upgrade-motor
SYS: Created new FixAtom. The current consequents are: Fix
hasAction upgrade-motor. Add another consequent?
USER: no
SYS: new ViolationFixRule created.

```

**Figure 4: The example protocol showing how the KA tool interacts with the user to build the rule defined in Table 2. As the user enters values, the KA tool creates the relevant instances in the PS ontology.**

with the (domain) enhancements made during the KA process, when it becomes ONT(lift', [diag, pnr]).

## 4. IMPLEMENTATION

We have implemented MAKTab as a plug-in for Protégé; it provides the functionality outlined above. By extending Protégé we are able to take advantage of its extensive import facilities. Further, it allows us and other PS developers to take advantage of the plug-ins which allow reasoning with various inference engines over an instantiated ontology, by providing rule converters for their chosen inference engine.

We have also implemented the two generic PSs discussed throughout the paper PS(pnr,-) and PS(diag,-) as sets of JessTab rules, based on pre-existing KBSs discussed previously. Both PSs have converters for translating the acquired rules into JessTab format. Further details will be provided in [4].

## 5. EVALUATION

We are evaluating MAKTab by using it with our generic propose-and-revise and diagnostic PSs to build related KBSs. Our initial evaluations have focused on the elevator domain; the developer has successfully built the KBS(pnr, lift) as outlined in Figure 1, and discussed throughout this paper, as well as KBS(diag, lift). The KBS(pnr, lift) contains all the rules described in the Sisyphus-2 specification, and produces the anticipated valid elevator configuration. We take this as a positive result that our tool can be used to build functioning, correct KBSs.

We are performing a usability study [9] on MAKTab's interface. In this evaluation we are using a small number of researchers to judge if the interface adheres to established usability criteria, based on the Xerox heuristic evaluation checklist [12]. We have also planned an evaluation of the tool by a number of knowledge engineers. For these experiments, subjects will be asked to build configuration and diagnosis KBSs in the computer hardware domain. We chose this domain as a basic system requires only a handful of relatively simple rules, while more advanced systems can contain many more, including relatively complex rules. This

affords greater flexibility in the KBSs the subjects are asked to build, than would be available in the lift domain, while still providing a thorough test of the system within an experimentally acceptable time period. Users are encouraged to use the “think out loud” approach, to allow us to determine where they have difficulties, and, after completion of the test they are also asked to complete a questionnaire in which they provide details/opinions about their experience of using the system. Initial results suggest people find the tool easy to use, with the guided KA tool in particular being very helpful when adding new rules.

## 6. CONCLUSION

In this paper we have presented a methodology for building KBSs by configuring reusable components to work together. Our methodology, and implementing tool, enable a user to reuse domain knowledge from a domain ontology, developed for a KBS which uses one type of PS, with other types of PSs to produce new KBSs. The first stage in this process involves mapping relevant domain knowledge to a further generic PS. In the second stage this knowledge is extended to provide the PS with the (additional) rules it requires to function within the domain. We have developed MAKTab, an extendable tool which supports our methodology. MAKTab can easily be extended to incorporate more types of mappings, better automatic mapping suggestions, and development of executable KBSs in languages other than JessTab. We believe this work moves the Knowledge Engineering community closer to fulfilling the dream of KBS creation by configuring reusable components, as MAKTab supports the reuse of existing domain knowledge with generic PSs and produces an KBS which can be executed in the same environment as MAKTab (Protégé). We are now planning further evaluations of the approach; first building KBSs in other domains, and then KBSs with other types of PSs.

## 7. ACKNOWLEDGMENTS

This work is supported under the Advanced Knowledge Technologies (AKT) IRC, which is sponsored by the UK EPSRC (grant GR/N15764/01). We are also grateful to the various developers on the Protégé team at Stanford University, Mark Musen who made available their version of the Sisyphus-VT code, and JessTab developer Henrik Eriksson, without which this work would have been significantly more challenging.

## 8. REFERENCES

- [1] J. Blythe, J. Kim, S. Ramachandran, and Y. Gil. An Integrated Environment for Knowledge Acquisition. In *Proceedings of the 2001 International Conference on Intelligent User Interfaces (IUI-2001)*, 2001.
- [2] J. Breuker and W. Van de Velde, editors. *CommonKADS Library for Expertise Modelling Reusable problem solving components*. IOS Press, 1994.
- [3] D. Corsar and D. Sleeman. Reusing JessTab rules in Protégé. *Knowledge-Based Systems*, 19(5):291–297, September 2006.
- [4] David Corsar. *KBS Development through Ontology Reuse and Ontology Driven Acquisition*. PhD thesis, forthcoming, University of Aberdeen, 2007.
- [5] M. Crubezy and M. Musen. Ontologies in Support of Problem Solving. In S. Staab, and R. Studer, editor, *Handbook on Ontologies in Information Systems*, chapter 16, pages 321–322. Springer, 2003.
- [6] J. Kingston. Pragmatic KADS: A methodological approach to a small knowledge based systems project. Technical report, Artificial Intelligence Applications Institute, University of Edinburgh, UK, November 1994. AIAI-TR-110, 1994.
- [7] S. Marcus, J. Stout, and J. McDermott. VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking. *AI Magazine*, 9(1):95–112, Spring 1988.
- [8] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, Fall 1991.
- [9] J. Nielsen and R. Molic. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, pages 249–256, New York, NY, USA, 1990. ACM Press.
- [10] B. Omelayenko, M. Crubezy, D. Fensel, R. Benjamins, B. Wielinga, E. Motta, M. Musen, and Y. Ding. *UPML: The Language and Tool Support for Making the Semantic Web Alive*, chapter 5, pages 141–170. *Spinning the Semantic Web*. The MIT Press, 2003.
- [11] J. Y. Park, J. Gennari, and M. Musen. Mappings for Reuse in Knowledge-Based Systems. In *11th Workshop on Knowledge Acquisition, Modelling and Management KAW 98*, 1998.
- [12] Deniese Pierotti. Heuristic evaluation - a system checklist. Xerox Corporation, 2000.
- [13] A. Th. Schreiber and W. P. Birmingham, editors. *International Journal of Human-Computer Studies*, volume 44. Elsevier Ltd, 1996.
- [14] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge Engineering and Management: the CommonKADS methodology*. MIT Press, December 1999.
- [15] M. Tallis. *A Script-Based Approach to Modifying Knowledge-Based Systems*. PhD thesis, University of Southern California, December 2000.