KBS Development on the (Semantic) Web

David Corsar and **Derek Sleeman**
Department of Computing Science, University of Aberdeen
Aberdeen, UK
{dcorsar, d.sleeman}@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. In the final section of this paper we speculate/discuss the issues involved in extending MAKTab so that it would be able to operate in the context of the (Semantic) Web. Here we introduce the idea of centralised mapping repositories.

Introduction

A Knowledge Based System (KBS) applies intelligent reasoning to a domain to solve a problem that would otherwise require considerable human time, effort and expertise. To achieve this, a KBS typically requires significant domain knowledge coupled with an intelligent reasoning module. The 1980's witnessed a change in the thinking behind KBSs development, with traditional time consuming techniques being replaced by more efficient, reuse based approaches. These new approaches revolved around the rapid configuration of reusable, independent components such as domain ontologies, problem solving methods (PSMs), problem solvers (PSs) and task specifications. To maximise reusability, every component was expressed generically: i.e. without reference to any other particular component; repositories were developed for each type of component and were responsible for the storage and provision of different examples of each of these components. In theory, to develop a new KBS, the knowledge engineer or an automated agent simply selects the appropriate components for their task (from repositories) and configures them to work together. Configuration was often simply viewed as a mapping process: for example, for a PSM to work with a particular domain ontology, mappings would be defined between the two: the mappings would allow the PSM to access and use the domain knowledge provided by the domain ontology during the PSM's execution. Despite considerable research focused on making this vision of KBS development a reality, it is, for various reasons, still yet to be fully realised. In summary, the various projects working to achieve this goal produced their own approaches, developing 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 failed were a) the lack of a standard formalism for representing domain ontologies, resulting in b) a lack of readily available domain ontologies, c) the lack of standards for representing rules which use the domain ontologies, and d) the lack of tools which allow standardised rule sets to operate over standardised representational schema.

Standards and technologies have subsequently progressed however, with formalisms such as OWL[1] providing a standard language for describing ontologies; SWRL[2] and RIF[3] providing languages for describing rules expressed against ontologies, and possibly, rule based PSs; and tools such as Protégé[4] providing 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 different ontology languages along with several reasoning plug-ins (called tabs) which allow various types of reasoning to be performed against an (instantiated) on-

[1] `http://www.w3.org/2004/OWL/`
[2] `http://www.w3.org/Submission/SWRL/`
[3] `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`
[4] `http://protege.stanford.edu`

tology. One of the most mature reasoning tabs is JessTab[5] which allows Jess[6] production rules to be executed against a KB (instantiated ontology). Berners-Lee *et al* (Berners-Lee, Hendler, & Lassila 2001) have also detailed a vision of the Semantic Web: a Web friendly to both humans and machines; where natural language text conveys knowledge to humans, and corresponding (instantiated) ontologies provide a form of easily accessible knowledge (a loosely structured KB) to machines: potentially providing a wealth of (domain) ontologies that can be exploited in the development of KBSs. Further, every ontology will be associated with rules describing how to map from it to other ontologies. It is likely that "off the shelf" tools will be used by authors to describe the page's content, usually against a standard domain ontology. This should provide a wealth of domain "Knowledge Bases (KB)s" (expressed against the same or similar ontologies), and, importantly, this will be associated with descriptions of how to map between the different (domain) ontologies.

As the Semantic Web vision becomes a reality, it will be desirable to make use of the wealth of new KBs that become available, for example by incorporating them into new KBSs. We have developed a KBS development methodology based on reuse, which is able to take advantage of the various advances in standards and technologies which have been made in recent years. We have previously reported our methodology and our supporting tool, MAKTab in (Corsar & Sleeman 2007). Briefly, our methodology for achieving KBS development through reuse consists of two phases. After selecting a generic PS and a domain ontology, the user maps relevant domain knowledge from the domain ontology to the target generic PS, typically providing it with knowledge of the concepts in the domain. This initial domain knowledge is then extended using a focused knowledge acquisition (KA) phase during which the user defines rules required by the PS for it to work in the chosen domain. In this paper we provide an outline of our methodology in MAKTab, the current support tool, and discuss applying the methodology on the (Semantic) Web.

This paper is structured as follows: first we discuss previous work on KBS development methodologies; we then discuss our approach to KBS development in the context of our original implementation, MAKTab; we then discuss conceptually how we plan to adapt it to work on the Semantic Web; and finally provide some conclusions.

Related Work

Various projects have looked at KBS development through configuration of reusable components: for example PSM Librarian, CommonKADS and IBROW3. IBROW3 is particularly relevant as it specifically focused on building KBSs through reuse of components on the Web. A good overview of the challenges faced with this type of KBS development is provided by Neches *et al* in (Neches *et al.* 1991). The Internet Reasoning Service 3 (IRS3) project, which provides a brokering service for building applications using Semantic Web Services is also relevant to building KBSs on the (Semantic) Web.

[5] `http://www.ida.liu.se/~her/JessTab/`

[6] `http://www.jessrules.com`

CommonKADS

CommonKADS (J. Breuker and W. Van de Velde 1994; Schreiber *et al.* 2000) 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. (Schreiber *et al.* 2000, chap. 6) provides outlines for 11 different 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.

PSM Librarian

PSM Librarian (Crubezy & Musen 2003) 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 (Omelayenko *et al.* 2003) is used to describe the available PSM libraries. The mapping ontology (Park, Gennari, & Musen 1998), 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.

IBROW3 Project

The main objective of the IBROW3[7] project was the development of an architecture that facilitated an "intelligent brokering 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 Internet Reasoning Service project.

Internet Reasoning Service 3 (IRS3)

The IRS3 project[8] is a further development of the IBROW3 work. The IRS3 project provides a semantic broker based approach to the development of applications from Semantic Web Services (Cabral *et al.* 2006), which automates the processes of mediating between a service requester (user) and one or more service providers (Semantic Web Services). Semantic Web Services describe the functionalities that they provide, in terms of the goals (tasks) that they fulfil. When provided with a task from a client, the IRS3 server uses its library of Semantic Web Services to determine appropriate services which can be used to achieve the task. The IRS3 server then manages the orchestration of these services, including any necessary communications between them (and handling any conceptual mismatches that can occur between different services), and their invocation, to create an new application for the user.

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 (Kingston 1994). 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, to provide any PS knowledge that the domain ontology is missing; and currently does not 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.

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 (Corsar & Sleeman 2006). 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.

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.* (Marcus, Stout, & McDermott 1988) developed the original system, SALT, and others, such as the Sisyphus-2 KA Challenge (Schreiber & Birmingham 1996) 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 elevator 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 proposed design with respect to the constraints, if OK then exit with success,

[7]`http://hcs.science.uva.nl/projects/IBROW3/home.html`

[8]`http://kmi.open.ac.uk/projects/irs`

3. If any constraints are violated, 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 subcomponents can be combined to form a complete system.
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, elevator) (see Table 1 for our notation), is composed of two components: an elevator domain ontology designed for propose-and-revise, ONT(elevator, [pnr]); and a propose-and-revise PS, PS(pnr, [elevator]). 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, [elevator]).

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, elevator), contains two components: a diagnostic elevator ontology, ONT(elevator, [diag]) specifying components, component malfunctions, symptoms and possible causes; and the diagnostic PS, PS(diag, [elevator]).

We have acquired a working version of both the KBS(pnr, elevator) and KBS(diag, elevator). Both systems were acquired as CLIPS[9] 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 so as to avoid being accused of designing them to work just within our framework.

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 elevator ontology, ONT(elevator, [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 elevator domain, KBS(pnr, elevator). Our algorithm for achieving this is to:

1. Split KBS(diag, elevator) into ONT(elevator, [diag]) and PS(diag, [elevator]) (this is easy in the Protégé/JessTab implementations).

[9]`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
elevator	Elevator domain
PS(pnr, -)	Domain independent pnr PS, which is composed of PS-ONT(pnr, -) and PS-RS(pnr)
PS(pnr, [elevator])	pnr PS developed in the context of the elevator domain, composed of components: PS-ONT(pnr, -), PS-RS(pnr), and PS-RS(pnr, [elevator])
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 *domain*, e.g. PS-RS(pnr, [elevator]) is the set of elevator 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, [elevator])	PS-ONT which defines the concepts used by PS-RS(pnr) and PS-RS(pnr, [elevator]) instantiated with relevant elevator knowledge (components and/or rules)
ONT(elevator)	Elevator domain ontology
ONT(elevator, [pnr])	Elevator domain ontology used by PS(pnr)
ONT(elevator', [pnr, diag])	Elevator domain ontology used by PS(pnr) and extended with knowledge for PS(diag)
KBS(pnr, elevator)	A KBS using the pnr PSM for the elevator domain. KBS(pnr, elevator) is composed of 2 linked components: ONT(elevator, [pnr]) and PS(pnr, [elevator])

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

2. Map <u>relevant</u> domain knowledge in ONT(elevator, [diag]) to PS-ONT(pnr, -) (extracted from PS(pnr, -)), to produce an initial PS-ONT(pnr, [elevator]).
3. Use PS-ONT(pnr, [elevator]) with the KA tool to acquire propose-and-revise rules for the elevator domain, to produce an extended PS-ONT(pnr, [elevator])).
4. Generate PS-RS(pnr, [elevator]) from PS-ONT(pnr, [elevator]).
5. Add any new domain concepts that are introduced in step 3 to ONT(elevator, [diag]) to create ONT(elevator', [diag, pnr]).
6. Combine PS(pnr, [elevator]) (which is composed of PS-RS(pnr, [elevator]) and PS-RS(pnr)) with ONT(elevator', [diag, pnr]) to create KBS(pnr, elevator).

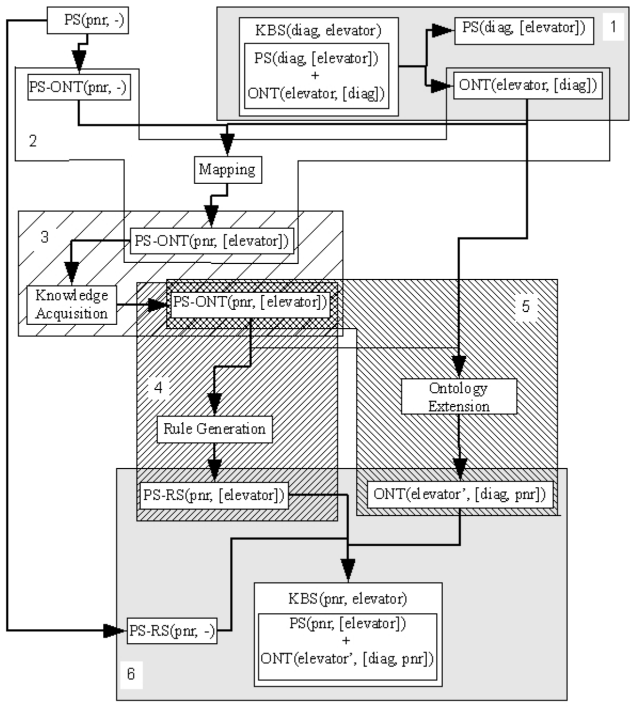

Figure 1: Outline architecture and algorithm for reusing the ONT(elevator, [diag]) from KBS(diag, elevator) with PS(pnr, -) to produce KBS(pnr, elevator).

Describing Generic PSs

Our methodology involves configuring a generic PSs to provide reasoning in a particular domain. To provide the user with maximum support during the configuration process the PS ontology provides relatively detailed descriptions of the required knowledge, which can be used by appropriate tools to support the user during configuration. The main support offered by the PS ontology is through describing the structure of domain knowledge, the types of domain specific rules that are used by the PS to work in a domain (in terms of the rule components and their structure), and various related meta-information. The KA stage uses these descriptions to acquire the rules from the user. We have developed a simple PS ontology, which developers can extend to provide descriptions for new types of generic PSs.

Our basic generic PS ontology includes classes for describing domain knowledge components, rules and meta-information about the PS and rules. Firstly, the `ProblemSolver` class, provides various pieces of information about the PS. 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 provides a description of the domain knowledge the PS requires. The basic domain knowledge descriptions consist of a `PSConcept` class, which has two subclasses: `SystemComponent` for representing the different components that will be used by the KBS (for example, the different elevator components, such as doors, motors, and cables); and `SystemVariable` for describing variables or parameters that will also be used by the KBS (for example, the maximum torque the elevator motor will experience, which varies depending on various component selections and other parameters). These classes can be extended and configured by developers to describe the domain knowledge requirements of their PSs.

The final purpose of the PS ontology is 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, the PS(pnr, -) contains three main types of rules: configuration rules, constraint rules, and fix rules. It also specifies that configuration rules, which can be used to define how to calculate a parameter value, are related to constraint rules, which can specify a constraint on that value. Constraint rules, in turn are related to fix rules, which specify what to do if that constraint is violated.

Rule descriptions are provided by extending various relevant SWRL classes, particularly `swrl:Imp`, `swrl:Atom`, and `swrl:AtomList`. Briefly, the types of antecedents and consequents that can be added to a rule are restricted to be relevant to the intended purpose of the rule. Specifically, to define the types of rules a PS uses to work in a domain, the developer creates a subclass of `swrl:Imp` for each rule type: placing appropriate constraints on the `swrl:body` (the rule antecedents) and `swrl:head` (the rule consequents) properties to restrict the lists of atoms (antecedents and consequents) to be of a particular `swrl:AtomList`. Similarly, subclasses of `swrl:AtomList` define lists which restrict the types of atoms (defined by subclasses of `swrl:Atom`) that can be added to that type of list. Again, appropriate subclasses of `swrl:Atom` constrain the type of knowledge that can be expressed to be appropriate to that particular type of atom. Using this approach, the developer can define a particular type of rule and ensure that rules of that type only contain appropriate information. For example, the constraint rule from PS-ONT(pnr, -), visualised in Figure 2, restricts the antecedents to be a list of constraints, and the consequents to be a list of constraint violations, ensuring all constraint rules follow the structure: IF *constraints not satisfied* THEN *assert violations*. An example rule using this schema is outlined in Table 2: this rule states that if the required horsepower is greater than the currently selected motor can provide, then assert a suitable violation of that constraint.

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

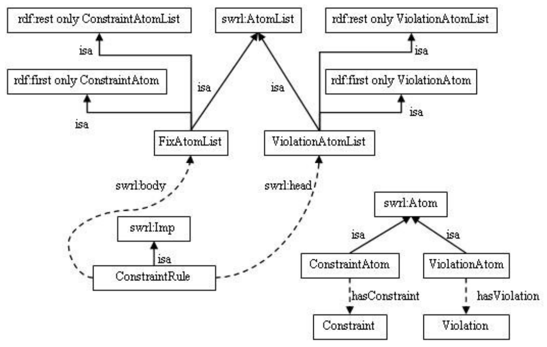

Figure 2: Visualisation of a Constraint rule from PS(pnr, -).

Instance Type	**Instance Name**	**Property Description**
`Constraint`	*`c1`*	`exp (required-horsepower > motor-horsepower)`
`Constraint-Atom`	*`ca`*	`hasConstraint` *`(c1)`*
`Constraint-AtomList`	*`cal`*	`rdf:first` *`(ca)`* `rdf:rest` *`(Nil)`*
`Violation`	*`v1`*	`hasConstraint` *`(c1)`*
`Violation-Atom`	*`va`*	`hasViolation` *`(v1)`*
`Violation-AtomList`	*`val`*	`rdf:first` *`(va)`* `rdf:rest` *`(Nil)`*
`Constraint-Rule`	*`cr1`*	`swrl:body` *`(cal)`* `swrl:head` *`(val)`*

Table 2: An instantiation of the ontology shown in Figure 2 which specifies the rule that if constraint c1 is violated then assert violation v1.

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 (domain) 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.* (Park, Gennari, & Musen 1998) for describing ontology mapping tools, as well as extensions for automatically suggesting mappings.

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(elevator, [pnr])) have the same symptoms, malfunctions and repairs, and should therefore be represented as one individual of the `PSConcept` class in PS(diag, [elevator]). 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.

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.

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.

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 PS class. This is necessary to allow, for example, many subclasses of the `Component` domain class (such as `Door`, `Motor`, etc.) in ONT(elevator, [pnr]) to be mapped to the single PS(diag, -) `Component` class. N:N mappings could be supported if required in the future.

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[10], over a user set value; secondly, matching those with a (user set) percentage of common constituents; and finally WordNet[11] suggests appropriate synonyms. This algorithm is based on that of PJMappingTab (Corsar & Sleeman 2006). We recognise that ontology mapping/matching is an active research field[12] 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

[10] `http://www.dcs.shef.ac.uk/~sam/simmetrics.html`

[11] `http://wordnet.princeton.edu/`

[12] See `http://www.ontologymatching.org` for details on ontology mapping research.

executed with several typical tests. At any stage the user is free to return and define/apply more mappings if necessary.

Focused Knowledge Acquisition

Having completed the mapping stage, a focused knowledge acquisition process is then used to extend the knowledge available to the target 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.

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[13]. This acquisition is based 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 `ConfigurationRule`, which describes how to calculate a value for a component of the elevator or a variable in the configuration.

For example, as shown in Figure 2, in the elevator domain, it is important that the motor has enough horsepower to produce enough torque to move the elevator. The horsepower required by the motor is dependent on the car capacity, car speed and the motor's system efficiency. If the motor can not provide enough horsepower, then an alternative motor with more horsepower should be used. These configuration, constraint and fix rules for the required motor horsepower parameter are illustrated in Figure 3 (`fixed width text` refers to domain concepts in PS ontology): **`ConfigRule-1`** checks if the `required-motor-horsepower` has already been calculated, if it has not, then the rule defines how the value should be calculated. **`ConstraintRule-1`** checks if the `required-motor-horsepower` is greater than the value of the `horsepower` property of the selected `motor`, if it is, the motor can not supply enough horsepower and an appropriate violation is asserted. Finally, **`FixRule-1`** defines that if more horsepower is required, then upgrade the motor by selecting one that can provide enough horsepower. Figure 4 provides an example protocol of the tool acquiring these rules (again, `fixed width text` refers to domain concepts in PS-ONT(pnr, [elevator])).

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

`ConfigRule-1`
IF `required-motor-horsepower` has no value THEN
`required-motor-horsepower` = (`car-capacity` * `car-speed` * 0.6)/33000 * `motor-system-efficiency`
`ConstraintRule-1`
IF `required-motor-horsepower` > `horsepower` of `motor` THEN
assert violation "need more horsepower"
`FixRule-1`
IF violation "need more horsepower" THEN
replace `motor` with another `motor` with `horserpower` > `required-motor-horsepower`

Figure 3: Example configuration, constraint, and fix rules for the PS(pnr,-) in the elevator domain.

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 (Corsar 2007).

KBS Development on the (Semantic) Web

Our current implementation, MAKTab, has been developed as a desktop application; for various reasons, it is desirable to provide a (Semantic) Web based tool for our KBS development methodology. Along with potentially providing access to many more ontologies, the Semantic Web will also provide details of how to map from one ontology to another, which can be used to enhance the KBS development process.

This potential application of our technique on the (Semantic) Web is outlined in Figure 5, and is described briefly below.[14]

1. Browsing the Semantic Web, the user finds a page(s) which provides the domain knowledge they wish to reason with.
2. The user provides the URL(s) of the selected web page(s), and our tool retrieves the associated ontology.
3. The user browses our library of generic PSs; selecting the one which provides the type of reasoning they wish to use.
4. The tool searches its repository of stored mappings for any previously used with the selected generic PS and the user's domain ontology; further, on the Semantic Web,

[14]Until the Semantic Web vision is realised, steps 1 and 2 can be substituted with the user providing their domain ontology directly to our tool.

SYS: Starting KA for `required-motor-horsepower`, creating a new configuration rule of the form "if *values1* then assign values *value2*"; please provide the definition of *values1*
USER: `required-motor-horsepower` has no value
SYS: Please provide the definition of *value2*
USER: `required-motor-horsepower = (car-capacity * car-speed *` 0.6)/33000 `* motor-system-efficiency`
SYS: Creating a new constraint rule of the form "if *values* then *violation*"; please provide the definition of *values*
USER: `required-motor-horsepower > horsepower` of `motor`
SYS: Please provide the definition of *violation*
USER: need more horsepower
SYS: Creating a new fix rule of the form "if *violation* then apply *fixes*", with *violation* defined as "need more horsepower" alter definition of *violation*?
USER: no
SYS: Please provide the definition of *fixes*
USER: select a new `motor` with `horsepower > required-motor-horsepower`

Figure 4: The example protocol showing how the KA tool interacts with the user to build the rule defined in Figure 3. As the user enters values, the KA tool will create the relevant individuals in the PS ontology (stored in the rule repository).

our tool will be able to use the mapping knowledge associated with the user's domain ontology (and others on the Semantic Web) to, if necessary, create a sequence of mappings which map the user's domain ontology to the generic PS through a series of intermediate ontologies (see below for details). After the user checks the mappings, altering them if necessary, the tool executes them, providing the generic PS with knowledge of some of the concepts in the domain. This step corresponds to step 2 in the MAKTab implementation described in Figure 1.

5. Using interactive Web technologies our tool supports the user with defining the required rules for each appropriate domain concept. New ontological concepts can be added to enhance the representation of the domain if necessary. This step corresponds to step 3 in the MAKTab implementation described in Figure 1.
6. Having defined the required rules, our tool generates an executable KBS by combining the generic PS code, the result of converting the user's defined rules into an executable format, and the enhanced user's domain ontology. This step corresponds to steps 4 and 6 in the MAKTab implementation described in Figure 1.
7. Our tool could then execute the KBS, returning the results to the user.

Domain Ontologies

In MAKTab domain ontologies are taken from a variety of sources including ontology search engines, repositories such as OntoSearch2 (Pan, Thomas, & Sleeman 2006), online directories and existing KBSs (if they can be decoupled from

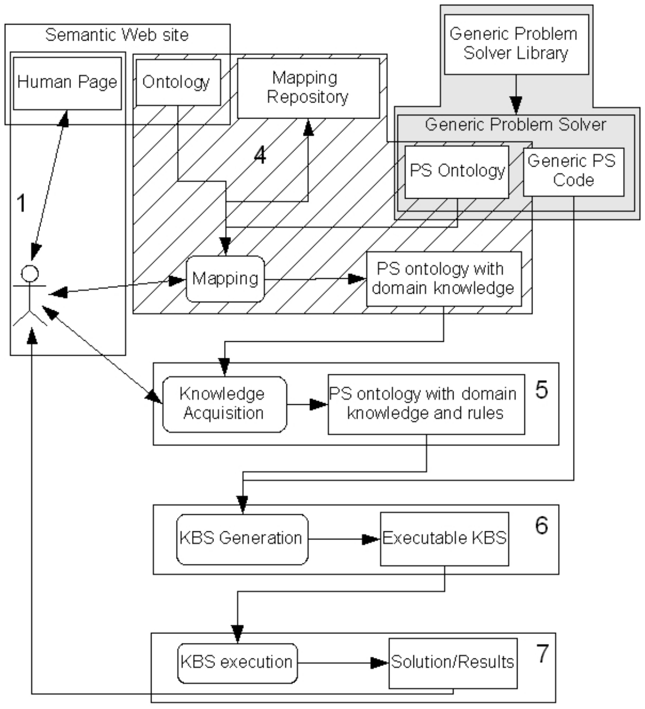

Figure 5: Building KBS on the Semantic Web.

the associated reasoning module). If the Semantic Web vision is fully realised, then ontologies will become much more widely available than they are today. Hopefully, this should mean that in the future, as well as existing sources, we will be able to import ontologies from appropriate Semantic Web Sites, which can then be used as the domain knowledge source for a new KBS. The quality of these ontologies, in terms of accuracy and completeness will, of course vary, but hopefully they will at least provide a loosely structured KB, which using our methodology can be further developed and extended into a suitable domain knowledge source for a KBS. We may also incorporate tools such as CleOn (Sleeman & Reul 2006) and RepairTab (Lam 2007) to detect and repair lexical and logical errors in the domain ontologies, to further improve their quality before they are used in KBS development.

Generic Problem Solvers

In MAKTab a generic PS is provided by the PS ontology, which is loaded into MAKTab before it is configured for a particular domain. MAKTab also handles the generation of the executable KBS. When applying our methodology on the Semantic Web, it will be desirable to store a repository of generic PSs, which can be used for browsing and selection. Ideally, providers of generic PSs will also handle the generation and execution of the final KBSs. We also anticipate that the generic PS stored in PS repositories will not just be implemented in JessTab, but in a wide range of programming languages. It will be important to ensure that these other programming languages can also successfully integrate with instantiated ontologies. Obvious candidates for consideration are CLIPS and Prolog, as there already exists Protégé plug-ins for these languages (CLIPSTab (Ameer 2003) and PrologTab[15]) which enable them to be used with an ontology in Protégé.

Mapping

As discussed previously, there are two main challenges for the user during the mapping stage: firstly determining which are the corresponding concepts in the domain ontology and the generic PS; and secondly determining how the correspondences between these concepts can be defined, these are likely to remain crucial steps in the Semantic Web version of the tool. In general, if one has a source (domain) concept (sc_1), and a target (PS) concept (tc_1) and a set of mapping rules M_1 ... M_n (possibly from a repository of mappings) describing how to map between many different concepts, then showing whether it is possible to find a set of mappings which will transform sc_1 to tc_1 is likely to be a sizeable search problem, where many potential configurations of mappings would need to be tried, many of which would lead to dead ends.

There is perhaps a more effective way of achieving mappings, which was implicit in (Berners-Lee, Hendler, & Lassila 2001). This paper specifies that the ontology representing a Semantic Web Site's content will be associated with a set of mappings between it and other ontologies (either for other Semantic Web Sites, or standard ontologies). Lets say a given page is represented by ontology O_1, which is associated with a set of mappings to change the concepts in O_1 to one of several ontologies, lets call them O_x and O_y. So effectively, O_1 is associated with a set of mappings from O_1 to O_x and O_1 to O_y; lets call these Mappings(1, x) and Mappings(1, y) respectively. If we wish to map to an ontology, O_2, representing another page's content, we would first check if Mappings(1, 2) is associated with O_1. If not, we could check if specified repositories of mappings have Mappings(x, 2) or Mappings(y, 2), in which case we would know we would be able to map from O_1 to O_2 in two stages. Of course, in general this mapping process could explore a larger number of mappings stages. Having mapping information centralised in mapping repositories would, in fact, allow the existence of a suitable sequence of mappings to be established relatively easily; the actual mapping process then simply requires the performance of the mappings as specified in the derived sequence.

Conclusions

We have developed a practical methodology for developing KBSs. Our methodology, and current 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 PS to produce a new KBS. We propose developing a (Semantic) Web based tool that supports the creation of KBSs from (reusable) components available on the (Semantic) Web; this tool will be derived from the operational non-Web based MAKTab. As the Semantic Web vision becomes reality, it should provide access to a wealth of new components suitable for building KBSs, particularly ontologies and mappings. These components have the potential to further automate and improve our tool for KBS development. We believe that this tool will be of particular benefit to the knowledge engineering community, and hopefully subsequently to domain experts.

[15]http://prologtab.sourceforge.net/

Acknowledgments

The MAKTab work was 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 the MAKTab work would have been significantly more challenging.

References

Ameer, R. 2003. Embedding CLIPS Engine in Protégé. In *Proceedings of Sixth International Protégé Workshop*.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American* May.

Cabral, L.; Domingue, J.; Galizia, S.; Gugliotta, A.; Norton, B.; Tanasuscu, V.; and Pedrinaci, C. 2006. IRS-III: A Broker for Semantic Web Services based Applications. In *The 5th International Semantic Web Conference (ISWC 2006)*.

Corsar, D., and Sleeman, D. 2006. Reusing JessTab rules in Protégé. *Knowledge-Based Systems* 19(5):291–297.

Corsar, D., and Sleeman, D. 2007. KBS Development through Ontology Mapping and Ontology Driven Acquisition. In Sleeman, D., and Barker, K., eds., *K-CAP '07: Proceedings of the 4th International Conference on Knowledge Capture*, 23–30. New York, NY, USA: ACM.

Corsar, D. 2007. *KBS Development through Ontology Reuse and Ontology Driven Acquisition*. Ph.D. Dissertation, forthcoming, University of Aberdeen.

Crubezy, M., and Musen, M. 2003. Ontologies in Support of Problem Solving. In S. Staab, and R. Studer., ed., *Handbook on Ontologies in Information Systems, International Handbooks on Information Systems*. Springer.

J. Breuker and W. Van de Velde., ed. 1994. *CommonKADS Library for Expertise Modelling Reusable problem solving components*. IOS Press.

Kingston, J. 1994. 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.

Lam, J. S. C. 2007. *Methods for Resolving Inconsistencies in Ontologies*. Ph.D. Dissertation, University of Aberdeen.

Marcus, S.; Stout, J.; and McDermott, J. 1988. VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking. *AI Magazine* 9(1):95–112.

Neches, R.; Fikes, R.; Finin, T.; Gruber, T.; Patil, R.; Senator, T.; and Swartout, W. R. 1991. Enabling technology for knowledge sharing. *AI Magazine* 12(3):36–56.

Omelayenko, B.; Crubezy, M.; Fensel, D.; Benjamins, R.; Wielinga, B.; Motta, E.; Musen, M.; and Ding, Y. 2003. *UPML: The Lanugage and Tool Support for Making the Semantic Web Alive*. Spinning the Semantic Web. The MIT Press. chapter 5, 141–170.

Pan, J. Z.; Thomas, E. J.; and Sleeman, D. H. 2006. ONTOSEARCH2: Searching and Querying Web Ontologies. In *IADIS International Conference WWW/Internet 2006 (University of Murcia)*, 211–219.

Park, J. Y.; Gennari, J.; and Musen, M. 1998. Mappings for Reuse in Knowledge-Based Systems. In *11th Workshop on Knowledge Acquisition, Modelling and Management KAW 98*.

Schreiber, A. T., and Birmingham, W. P., eds. 1996. *International Journal of Human-Computer Studies*, volume 44. Elsevier Ltd.

Schreiber, G.; de Hoog, R.; Akkermans, H.; Anjewierden, A.; Shadbolt, N.; and de Velde, W. V., eds. 2000. *Knowledge Engineering and Management: the CommonKADS methodology*. MIT Pressss.

Sleeman, D. H., and Reul, Q. H. 2006. CleanONTO: Evaluating Taxonomic Relationships in Ontologies. In Vrandecic, D.; Surez-Figueroa, M. C.; Gangemi, A.; and Sure, Y., eds., *Proceedings of 4th International EON Workshop on Evaluation of Ontologies for the Web*.