

Developing Knowledge-Based Systems using the Semantic Web^a

^aPresented at SSS-08 on Symbiotic Relationships between the Semantic Web & Knowledge Engineering

David Corsar & 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 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 knowledge-based systems: 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 knowledge-based systems 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 knowledge-based systems 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 knowledge-based systems which is executable. In the final section of this paper we discuss the issues involved in extending MAKTab so that it would be able to operate in the context of the (Semantic) Web. Here we use the idea of centralised mapping repositories and mapping composition. This work contributes to the vision of the Web, which contains components (both problem solvers and instantiated ontologies (knowledge bases)) that tools (like MAKTab) can use to create knowledge-based systems which subsequently can enhance the richness of the Web by providing yet further knowledge-based Web-services.

Keywords: Knowledge-Based Systems, Ontology Mapping, Knowledge Acquisition

1. 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. Recent approaches to KBS development typically revolve 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 is expressed generically: i.e. without reference to any other particular component; further, repositories of these components are now becoming available. 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 has often been 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 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 the primary reasons why previous approaches failed were the lack of a standard formalism for representing domain ontologies, which resulted in a lack of readily available domain ontologies. Further, there was a lack of standards for representing procedural knowledge, and a lack of tools which allowed standardised rule sets to operate over standardised domain knowledge formalisms.

Standards and technologies have subsequently progressed however, with formalisms such as the Web Ontology Language (OWL)¹ providing a standard language for describing ontologies; the Semantic Web Rule Language (SWRL)² and the Rule Interchange Format³ providing languages for describing rules expressed against ontologies, and possibly, rule based PSs; and tools such as Protégé⁴ providing a mature, extendable framework for both creating and using ontologies. In fact, Protégé provides a good environment for reuse-based KBS development as it includes extensive ontology import facilities, along with several reasoning plug-ins (called tabs) which allow various types of reasoning to be performed against an (instantiated) ontology. One of the most mature reasoning tabs is JessTab⁵ which allows Jess⁶ production rules to be executed against a KB (instantiated ontology).

As the Semantic Web vision [1] 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 [4]. 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.

In [1], Berners-Lee *et al* describe 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. This work contributes to the vision of the Web, which contains components (both PSs and instantiated ontologies (KBs)) that tools (like MAKTab) can use to create KBSs which subsequently can enhance the richness of the Web by providing yet further knowledge-based Web-services.

This paper is structured as follows: first we discuss previous work on reuse of PSs and knowledge-based components; 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.

2. RELATED WORK

Various projects have looked at KBS development through the configuration of reusable components: for example PSM Librarian, CommonKADS and IBROW3. IBROW3 is particularly

¹<http://www.w3.org/2004/DWL/>

²<http://www.w3.org/Submission/SWRL/>

³http://www.w3.org/2005/rules/wiki/RIF_Working_Group

⁴<http://protege.stanford.edu>

⁵<http://www.ida.liu.se/~her/JessTab/>

⁶<http://www.jessrules.com>

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 [11]. The Internet Reasoning Service 3 project, which provides a brokering service for building applications using Semantic Web Services is also relevant to building KBSs on the (Semantic) Web.

2.1. CommonKADS

CommonKADS [15] 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. Developing a KBS using this methodology involves modelling the various aspects of the organisation (in which the KBS will be deployed) that are relevant to the task the KBS will be performing. Relevant aspects include the agents who currently perform the task, the knowledge that is required to perform the task, and how the task is currently performed. [15] provides the developer with guidance regarding how these models may be created, but it is left to the developer to implement and “assemble” them into a working system.

2.2. PSM Librarian

PSM Librarian [6] provides a KBS development methodology based on the reuse and configuration of domain ontologies and problem solving knowledge. In this methodology the requirements of a PSM are described by an ontology; developing a new KBS involves defining mappings between a domain ontology and a PSM’s requirements. Once defined, these mappings enable the PSM to access and reason with the domain knowledge. There is currently no support for executing the configured KBS however.

2.3. IBROW3 Project

The main objective of the IBROW3⁷ 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.” The UPML (Unified Problem-Solving Method Development Language) meta-ontology [12] 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 project.

2.4. Internet Reasoning Service 3 (IRS3)

The IRS3 project⁸ 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 [2], 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 communication between them (and handling any conceptual mismatches that can occur between different services), and their invocation, to create an new application for the user.

⁷<http://hcs.science.uva.nl/projects/IBROW3/home.html>

⁸<http://kmi.open.ac.uk/projects/irs>

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 requires considerable documentation, which can add substantial overheads to the KBS development project [8]. Further, due to lack of good quality support tools, the CommonKADS methodology provides the developer with only minimal support with the difficult task of developing and assembling these models.

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 domain knowledge from sources other than the domain ontology; 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.

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 implemented our methodology with MAKTab, a plug-in for the Protégé environment. 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 support the user when extending the generic PS to their chosen domain.

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.* [10] developed the original system, SALT, and others, such as the Sisyphus-2 KA Challenge [14] 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 meet 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,
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.

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 (lift in British English)
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 <i>domain</i> , 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])	ONT(elevator, [pnr]) extended with knowledge for PS(diag)
KBS(pnr, elevator)	A KBS using the pnr PSM for elevator domain, 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.

So from this perspective, a KBS is composed of domain knowledge and problem solving knowledge. For example, the KBS described above, henceforth referred to as KBS(pnr, elevator)⁹¹⁰ (that is, a KBS which uses a propose-and-revise PS developed in the context of the elevator domain), is composed of two components: an elevator domain ontology designed for propose-and-revise, ONT(elevator, [pnr]); and a propose-and-revise PS which is developed in the context of the elevator domain, PS(pnr, [elevator]). The latter is defined as a rule set which captures the generic propose-and-revise algorithm (i.e. does not contain any domain specific knowledge) (PS-RS(pnr, -)), an ontology to capture the essential components of the propose-and-revise algorithm (i.e. the constraints, the fixes, etc.), without reference to any domain, namely PS-ONT(pnr, -), and (an implementation of the) domain (elevator) specific propose-and-revise 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) (that is, a KBS which uses a diagnostic PS in the context of the elevator domain) contains two components: a diagnostic elevator ontology, ONT(elevator, [diag]) specifying components, malfunctions, symptoms and possible causes; and the diagnostic PS developed in the context of the elevator domain, PS(diag, [elevator]).

We have acquired a working version of both the KBS(pnr, elevator) and KBS(diag, elevator). Both systems were acquired as CLIPS¹¹ 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 careful not to structurally alter their domain and PS ontologies. Details of the re-engineering process are available in [5].

⁹see Table 1 for our notation

¹⁰In our notation, brackets are only used to improve readability.

¹¹<http://www.glg.net/clips/CLIPS.html>

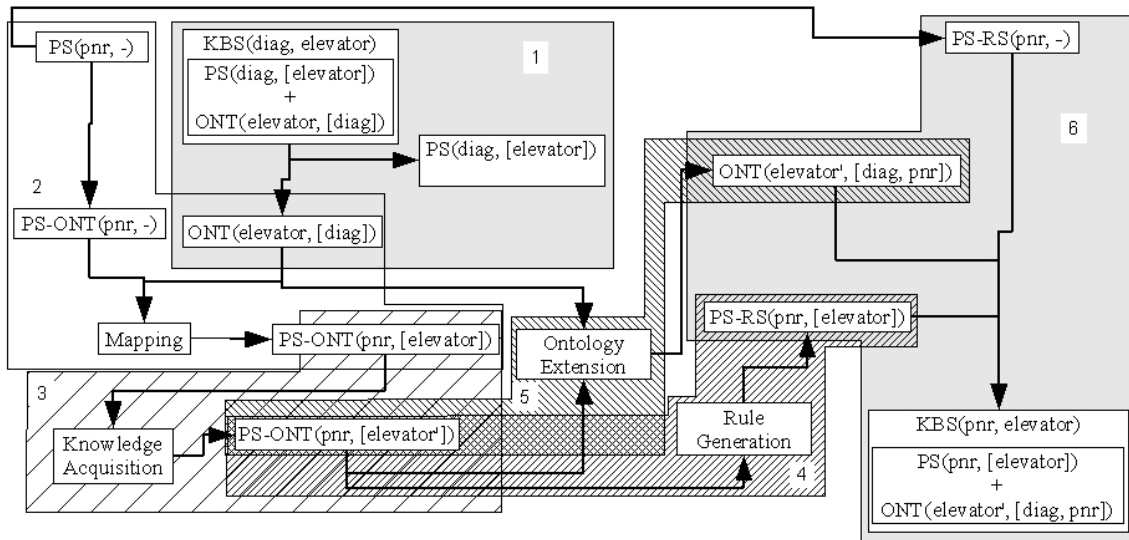


FIGURE 1: Outline architecture and algorithm for reusing the ONT(elevator, [diag]) from KBS(diag, elevator) with PS(pnr, -) to produce KBS(pnr, elevator).

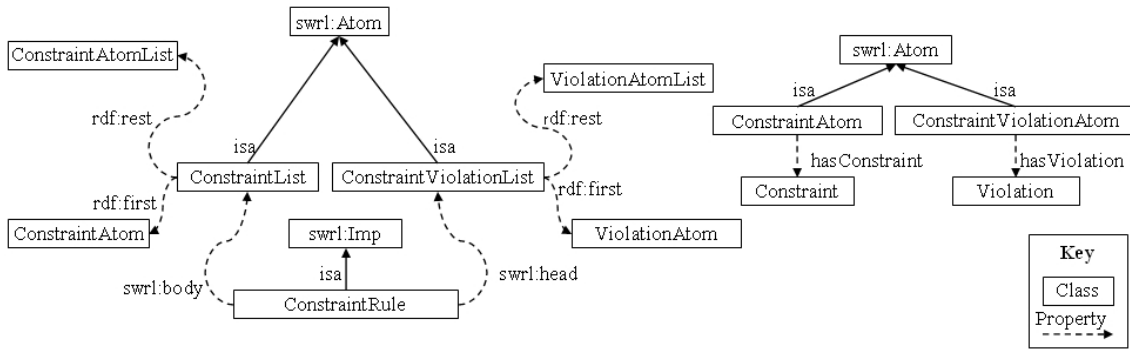
Our methodology is such that the user should be able to extract the domain ontology from an existing KBS and rapidly configure a further generic PS to work with it to produce a new KBS. Figure 1 illustrates one such example in which a diagnostic elevator ontology, ONT(elevator, [diag]) (extracted from 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 relatively easy in the Protégé/JessTab implementations).
2. Map relevant 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).

3.2. 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 PS ontology provides descriptions of 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. The main class for describing domain knowledge is PSConcept, which has two subclasses: SystemComponent for representing the different components that will be used by the KBS (for example, the different elevator components,


FIGURE 2: Visualisation of the Constraint rule from PS(pnr, -).

Instance Type	Instance Name	Property Description
Constraint	<i>c1</i>	exp (required-horsepower > motor-horsepower)
ConstraintAtom	<i>ca</i>	hasConstraint (<i>c1</i>)
ConstraintAtomList	<i>cal</i>	rdf:first (<i>ca</i>) rdf:rest (<i>Nil</i>)
Violation	<i>v1</i>	hasConstraint (<i>c1</i>)
ViolationAtom	<i>va</i>	hasViolation (<i>v1</i>)
ViolationAtomList	<i>val</i>	rdf:first (<i>va</i>) rdf:rest (<i>Nil</i>)
ConstraintRule	<i>cr1</i>	swrl:body (<i>cal</i>) swrl:head (<i>val</i>)

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*.

such as doors, motors, and cables); and `SystemVariable` for describing variables or parameters that will also be used.

The rules are described by extending various 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 antecedents) and `swrl:head` (the consequents) properties to restrict the lists of atoms (antecedents and consequents) to be of a particular `swrl:AtomList` subclass. 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 generic 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, which states that if the required horsepower is greater than the currently selected motor can provide, then assert a violation of that constraint.

Our generic PS ontology also contains two main classes for providing meta-information about the PS, which is used by MAKTab. The `ProblemSolver` class, which provides various pieces of information about the PS, such as textual descriptions of the PS, how it can be customised, which rules MAKTab should start the KA process with, and an implementation of any generic PS rules and functions. Additionally, for use during the KA stage the `RuleMetaClass` class provides an textual description of a rule's purpose, and specifies which rule-types the rule is related to.

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 (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 to meet future mapping requirements.

We currently provide a range of mapping types which have proved useful in our experiments so far; however, 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, so new mapping types can be easily incorporated as needed. Currently, the tool supports N:1 mappings, allowing multiple domain classes to map to one PS class. To reduce the number of mappings that must be created, when defining a mapping for a class, the user can specify if it should also be applied to that class's subclasses, and if so, how deep it should be applied.

To further reduce the number of mappings the user is required to provide, MAKTab suggests mappings by attempting to match class and property names in the domain ontology with those in the PS. The algorithm for performing this is based on that of PJMappingTab [3]. We recognise that ontology mapping/matching is an active research field¹² and so we have designed the mapping 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 user can, at any stage of the development process, define/apply more mappings.

3.4. Focused Knowledge Acquisition

Having completed the mapping stage, a focused knowledge acquisition (KA) process is then used to extend the knowledge available to the target PS. The KA tool of MAKTab uses the information provided in the PS ontology, described above, to guide the acquisition of the domain specific rules¹³ which the PS requires to function in the chosen domain. 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). Currently the KA process interacts with a human user who is *assumed* to be capable of providing the required information. 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.

For example, as shown in Table 2, in the elevator domain, it is important that the motor has sufficient 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, an alternative more powerful motor 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 it is calculated as defined by the rule. `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

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

¹³If 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 and assuming if the type of one rule's consequents matches another rule's antecedents, then they are 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 horsepower > required-motor-horsepower
    
```

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

```

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.

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])).

Once these rules have been defined by the user, MAKTab then converts them into a form that can be executed by an appropriate inference engine.

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 existing plug-ins which enable a number of inference engines (for example, Prolog and CLIPS) to reason over existing KBs (instantiated ontologies) by producing appropriate rule converters.

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. We have also developed rule converters for both PSs which translate the acquired rules into JessTab format, [5].

5. EVALUATION

We have performed three evaluations of MAKTab, namely developer evaluation, user evaluation, and interface evaluation. These evaluations and their results are discussed in [5].

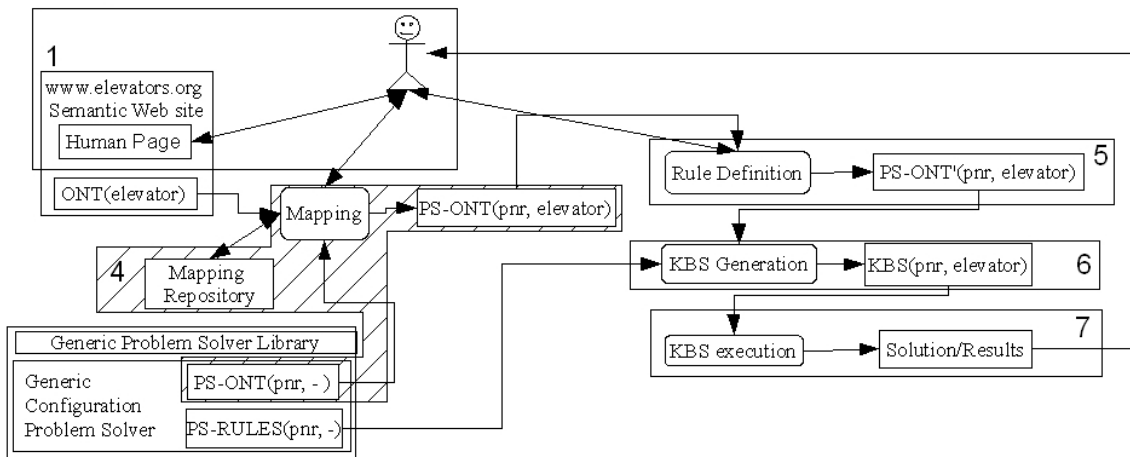


FIGURE 5: Building KBS on the Semantic Web.

6. 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 page(s) which provide 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 existing ontology associated with that page.
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, 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.

6.1. Domain Ontologies

In MAKTab, domain ontologies are taken from a variety of sources including ontology search engines, repositories such as OntoSearch2 [13], online directories and existing KBSs (if they can

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

be decoupled from 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, we will also 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 [16] and RepairTab [9] to detect and repair lexical and logical errors in the domain ontologies, to further improve their quality before they are used in KBS development.

6.2. 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. 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.

6.3. 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 $M_1 \dots M_n$ (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.

The idea of mapping composition has received attention in the database community [7], and is implicit in [1] as a way of determining mappings. This paper ([1]) specifies that the ontology representing a Semantic Web site's content should be associated with a set of mappings between it and other ontologies (either to other Semantic Web sites, or standard ontologies). Let's 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 stages. Having this information centralised in mapping repositories would, in fact, allow the existence of a suitable composition 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.

7. CONCLUSIONS

We have developed a practical methodology for developing KBSs. Our methodology, and associated tool, enable a user to reuse 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, mappings, and problem solvers. 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. We are confident that these activities will result in a significant number of additional knowledge-based Web-services.

8. ACKNOWLEDGMENTS

The MAKTab work was supported under the AKT IRC, which was sponsored by the UK EPSRC (grant GR/N15764/01). We are also grateful to the Protégé team at Stanford University, Mark Musen who made available their version of the Sisyphus-2 code, and JessTab developer Henrik Eriksson, without which the MAKTab work would have been significantly more challenging.

REFERENCES

- [1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May, 2001.
- [2] L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, B. Norton, V. Tanasuscu, and C. Pedrinaci. IRS-III: A Broker for Semantic Web Services based Applications. In *The 5th International Semantic Web Conference (ISWC 2006)*, 2006.
- [3] D. Corsar and D. Sleeman. Reusing JessTab rules in Protégé. *Knowledge-Based Systems*, 19(5):291–297, September 2006.
- [4] D. Corsar and D. Sleeman. KBS Development through Ontology Mapping and Ontology Driven Acquisition. In D. Sleeman and K. Barker, editors, *K-CAP '07: Proceedings of the 4th International Conference on Knowledge Capture*, pages 23–30, New York, USA, 2007. ACM.
- [5] David Corsar. *KBS Development through Ontology Reuse and Ontology Driven Acquisition*. PhD thesis, forthcoming, University of Aberdeen.
- [6] M. Crubezy and M. Musen. Ontologies in Support of Problem Solving. In S. Staab, and R. Studer, editor, *Handbook on Ontologies in Information Systems, International Handbooks on Information Systems*. Springer, 2003.
- [7] P. Cudré-Mauroux, K. Aberer (Eds), A. I. Abdelmoty, T. Catarci, E. Damiani, A. Illaramendi, M. Jarrar, R. Meersman, E. J. Neuhold, C. Parent, K.-U. Sattler, M. Scannapieco, S. Spaccapietra, P. Spyns, and G. De Tré. Viewpoints on emergent semantics. *Journal on Data Semantics*, VI:1–27, 2006.
- [8] 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.
- [9] Joey Sik Chun Lam. *Methods for Resolving Inconsistencies in Ontologies*. PhD thesis, University of Aberdeen, 2007.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] J. Z. Pan, E. J. Thomas, and D. H. Sleeman. ONTOSEARCH2: Searching and Querying Web Ontologies. In *IADIS International Conference WWW/Internet 2006 (University of Murcia)*, pages 211–219, 2006.
- [14] A. Th. Schreiber and W. P. Birmingham, editors. *International Journal of Human-Computer Studies*, volume 44. Elsevier Ltd, 1996.
- [15] G. Schreiber, R. de Hoog, H. Akkermans, A. Anjewierden, N. Shadbolt, and W. Van de Velde, editors. *Knowledge Engineering and Management: the CommonKADS methodology*. MIT Press, 2000.
- [16] D. H. Sleeman and Q. H. Reul. CleanONTO: Evaluating Taxonomic Relationships in Ontologies. In D. Vrandečić, M. C. Surez-Figueroa, A. Gangemi, and Y. Sure, editors, *Proc. of 4th International EON Workshop on Evaluation of Ontologies for the Web*, 2006.