Semantic Web Reasoning using a Blackboard System

Craig McKenzie, Alun Preece, and Peter Gray

University of Aberdeen, Department of Computing Science
Aberdeen AB24 3UE, UK
{cmckenzie,apreece,pgray}@csd.abdn.ac.uk

Abstract. In this paper, we discuss the need for a hybrid reasoning approach to handing Semantic Web (SW) data and explain why we believe that the Blackboard Architecture is particularly suitable. We describe how we have utilised it for coordinating a combination of ontological inference, rules and constraint based reasoning within a SW context.
After describing the metaphor on which the Blackboard Architecture is based we introduce its key components: the blackboard Panels containing the solution space facts and problem related goals and sub-goals; the differing behaviours of the associated Knowledge Sources and how they interact with the blackboard; and, finally, the Controller and how it manages and focuses the problem solving effort.
To help clarify, we use our test-bed system, the AKTive Workgroup Builder and Blackboard (AWB+B) to explain some of the issues and problems encountered when implementing a SW Blackboard System in a problem oriented context.

1 Introduction & Motivation

The W3C Semantic Web Activity Group[1] describes the Semantic Web (SW) as providing *"...a common framework that allows data to be shared and reused across application, enterprise, and community boundaries."* Unfortunately, since this machine processable information is essentially a "symbolic" version of the current web, the drawback is that there is no regulation over the proffered content, creating many complicating factors and making the task of utilising (and reasoning against) "open web" data far from trivial. Because the Logic Layer of the SW architecture means not only the use of logic to enrich data but also the application of logic to "do something" with the data [12], our research interest lies in exploring the suitability of a Blackboard System to utilise incomplete, SW information in a closed world, problem oriented context, i.e. using SW data to create a (finite domain) Constraint Satisfaction Problem (CSP) before attempting to solve it.

An interesting starting domain was within the context of the CS AKTive Space[2] [16], namely the Computing Science (CS) community in the UK. Our

---

[1] http://www.w3.org/2001/sw/

[2] http://cs.aktivespace.org

demo application, the AKTive Workgroup Builder and Blackboard (AWB+B), is a SW application that attempts to construct one or more working groups of people from a pool of known individuals. Workgroup composition must adhere to a set of user defined constraints, e.g. "the workgroup must contain between 5 and 10 individuals" or "at least half the members of the workgroup must have a research interest of *Agents*".

Since our problem combines ontological inference, rules and constraint based reasoning, we believe that a combination of reasoning methods are necessary. The "one size fits all" reasoning theory was questioned in [17] when a DL based reasoner was compared to a First-Order prover. The final conclusion was that when dealing with a very expressive OWL DL ontology a combination of both is necessary because there was no known single reasoning algorithm able to adequately cope with the full expressivity possible with the OWL DL language. They also flagged slow performance speed as a potential hurdle. Therefore, for this to be efficient, a hybrid reasoning [2] approach is required.

Once this necessity for hybrid reasoning was identified, we realised that there is nothing in the architecture of the Semantic Web for coordinating this effort. We believe the Blackboard architecture is appropriate as it meets our requirements – supporting the use of distributed Knowledge Sources (KSs) responding to a central, shared knowledge base via a control mechanism [15, 3].

The paper is organised as follows: Section 2 introduces our test-bed application, the AWB+B, and explains the process of building workgroups; Section 3 describes the blackboard analogy before comparing the traditional approach to our Semantic Web based approach; Section 4 describes the role of the Knowledge Sources and discusses their individual attributes; Section 5 describes the controlling mechanism of the blackboard; Section 6 describes the planned direction of our future work; and Section 7 provides discussion and our conclusions.

2   Building Workgroups

The AKTive Workgroup Builder and Blackboard (AWB+B) is a new incarnation of our earlier version of the system (AWB [13]) that did not use the blackboard architecture. Like its predecessor, the AWB+B is a web-based application that tackles the problem of assembling a workshop containing one or more workgroups from a pool of known people. Since the user is not expected to have knowledge about the lower level operations of the blackboard, we assume that all the necessary RDF information resources (describing the people, constraints, derivation rules, etc) to be included are known to the user (via URIs). This allows the blackboard to be initialised and the KSs to be dynamically created and registered with the blackboard "behind the scenes".

The RDF data processed by the AWB+B contains information about each individual's research interests, publications and projects they have been involved in. The detail of this information will vary depending upon what is published by a particular data source. Ideally, this information will need to be reasoned against in order to infer additional facts that may not have been explicitly stated

– for example projects that a person has worked on or papers that they have published can imply additional research interests.

3   The Blackboard Analogy

The concept of a Blackboard System is based upon a metaphor whereby a group of people, each with differing expertise and knowledge, are all standing around a blackboard deliberating over a problem that has been written up on it. Everyone understands that the ultimate goal is to solve the problem and that they will know the solution when they see it but, at this point in time, no single individual can derive the final solution on their own. The process begins when one person looks at the problem description on the board and realises that he/she can make a small relevant contribution. They write their finding onto the blackboard for the others to see. This inspires another person to a further idea, which they also write on the blackboard. This scenario continues until eventually a solution is reached via these incremental, cooperative steps (for a fuller description see [15]). No-one is allowed to communicate directly, everything must be done through the blackboard which becomes a shared "thinking space" for all the participants. We must also consider the protocol of how everyone writes on the blackboard. For example, if there is only one piece of chalk, how is the decision made as to whom gets to use it (and when) to write on the blackboard? Potentially, this could be by having someone act as a *controller*. In computing terms, the architecture of a Blackboard System has the "blackboard" as a shared Knowledge Base, and the "people" as various Knowledge Sources – we discuss KSs in more detail in Section 4.

3.1   Traditional Approach

The pioneering blackboard systems (Hearsay-II [6], HASP/SIAP [8], CRYSALIS [5] and OPM [11]) maintained the blackboard as a shared data repository representing a communal work area or "solution space" of potential solution components. The associated KSs were able to view the contents of the blackboard and react by indicating what they could contribute. They were only allowed to modify the contents of the blackboard if/when requested to do so by the Controller. For this to work efficiently, the data held on the blackboard must be structured hierarchically into Abstraction Levels (see Figure 1); multiple distinct hierarchies were referred to as Panels.

This organisation served two purposes. Firstly, it aided each KS to check if it can contribute (i.e. the KS was activated, or *triggered*, by the propagation of information onto an abstraction level that it was monitoring). Secondly, it helped focus the search for the solution. As the name suggests, each layer is an abstraction using concepts that hide the detail on the layer below it. To clarify, using the domain of speech understanding, suppose the lowest abstraction level could be the phonetic sounds accepted by the system; the level above could be potential combinations of these sounds into letter groups; the next level being
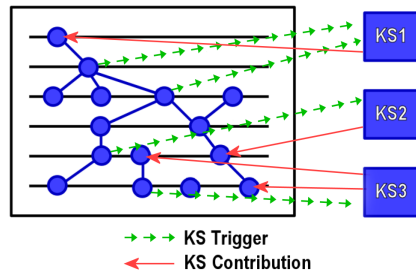
Fig. 1. Each Knowledge Source (KS) can view the Abstraction Levels within a Blackboard Panel. A KS can be *triggered* by any of the items on blackboard, allowing it to *contribute* something at any of the abstraction levels.
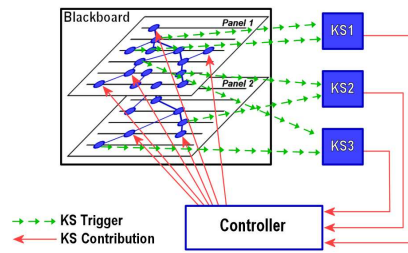
Fig. 2. The core architectural components of a Blackboard System. Each KS can view the contents of the Blackboard Panels, but it is the Controller that decides which KS(s) are allowed to contribute to the Blackboard.

single words; the next level could be phrases; with, finally, the topmost level consisting of potential whole sentences. A word-dictionary KS would examine the phonetic letter groups and combine these to form words, which (controller permitting) it would then post onto the level above.

The nature of each abstraction level and the actual entries within each level, can vary from implementation to implementation depending upon the nature of the problem attempted. Instead of the bottom-up approach used in the example, a top-down approach may be required, so the first abstraction level is vague with later ones becoming more refined. Likewise a KS's trigger could span multiple layers with a contribution also affecting one or more layers (see Figure 2).

As mentioned already, the decision of what is (or is not) placed on the blackboard is made by the controller, and the complexity of the solving strategy adopted can vary from a simplistic "just action everything" approach to a more complex goal driven algorithm. The key point is that it directs the solving process, via goals and sub-goals, that each of the KSs can be triggered by. This also helps to ensure that only *relevant* information is added. Since the triggering action can be dependent upon information added by a different KS, this results in an opportunistic solving paradigm. A blackboard system is fundamentally backward chaining – it is goal driven. In our case, the initial goal placed on the blackboard is to find a solution to a specified workgroup problem.

3.2 Semantic Web Approach

Our *Semantic Web Blackboard* maintains all the principles of the *traditional* blackboards but improves upon them by incorporating some of the concepts of the Semantic Web. The notion of Abstraction Levels aligns itself well to the hierarchical, structured nature of an ontology. In our test-bed system, AWB+B (discussed later), the information represented on blackboard is stored as an RDF

graph. This also has the advantage that the contents of the blackboard can be easily serialised into textual form – either for debugging purposes or propagation of the contents – and represented in a well known and understood format (e.g. RDF, N3).

To the best of our knowledge, in the past the blackboard has always been passive with any deductive mechanism placed in the KSs. While not wishing to stray too far from the original concepts of the architecture, we decided to introduce an element of intelligence to the blackboard itself by enabling it to perform reasoning on the class hierarchy being added to it. Here we are only materialising all the transitive sub-class/property relations and all the instance type relations. For example, if a class C_1 is defined as being a sub-class of C_2 and C_2 is a sub-class of C_3 then the blackboard would assert that C_1 is a sub-class of C_3. The blackboard also has the ability to assert new `<rdf:type>` statements about individuals. Continuing the previous example, if X is an instance of C_1 and C_1 is a sub-class of C_2 then we can assert that X is also an instance of C_2.

We elected to only perform this type of reasoning and not a richer type of classification that is possible within OWL (e.g. using property domain and ranges) since this is such a common operation that having it done by the blackboard eliminates the need for frequent call outs to KS that would perform the same function. Unfortunately, enabling the Blackboard to make inferences about itself must be treated with caution. Since reasoning is both difficult and time consuming, it would be undesirable if the actual blackboard became a bottleneck while it attempted to fully reason about itself and denied all KSs from contributing – hence we have not increased the blackboard's inference ability any further. The problem is that there is absolutely no guarantee of decidability w.r.t. information placed on the blackboard. In ontological terms, a KS could contribute triples that make the blackboard contents OWL Full (meaning that it could not be fully reasoned against anyway). To prevent this, we inhibit the statements placed by the KSs to be OWL Lite. We can guarantee decidability by ensuring all contributed statements are based upon a known URIs which can be checked to classify its OWL species. This can be done in one of two ways: the underlying ontology is checked against a register of known OWL Lite Ontologies; or, providing it is not too computationally expensive, the OWL species can be checked with an existing validator (e.g. Pellet[3], WonderWeb[4], BBN[5]).

4 Behaviours of Knowledge Sources

The KSs represent the problem solving knowledge of the system. Each KS can be regarded as being an *independent* domain expert with information *relevant* to the problem at hand. The key point is that no assumptions should be made about the capabilities of a KS – conceptually it should be regarded as a black box. Due to the tightly coupled nature of the KSs and the Blackboard, all KSs

---

[3] `http://www.mindswap.org/2003/pellet/species.shtml`
[4] `http://phoebus.cs.man.ac.uk:9999/OWL/Validator`
[5] `http://owl.bbn.com/validator/`

must be "registered" so that they can view the blackboard contents and inform the Controller of any potential contributions.

KSs can access the blackboard and continually check to see if they can contribute. Each one has a precondition (or event trigger) and an action (*what* it can add to the blackboard). The blackboard is monotonic, facts are only ever added by the KSs, never retracted. This mechanism is usually overseen by a controller that monitors changes to the blackboard and delegates actions accordingly. The whole process is driven by the posting of goals which a KS either offers a direct solution to, or breaks down further into sub-goals, indicating that more knowledge is required.

The following sub-sections describe the main types of KS currently implemented within the AWB+B system based on their behaviours w.r.t. the blackboard. This is by no means an exhaustive list of all the possible types of KS and it should be noted that future KSs could combine some of these behaviours, but we have not explored this yet. Since our interest lies in rule and constraint based reasoning, we discuss the KSs relating to these areas in greater depth.

4.1 User (Human) KS

While this may not be immediately obvious, the user of the system can be regarded as a type of KS. This represents "human" knowledge which is entered via the web-based user interface. In AWB+B terms this would be a user specifying the problem parameters, e.g. the number of workgroups to be built, the size of each workgroup, any associated compositional constraints etc. Once all the necessary information for the CSP has been entered, the KS transforms it into the starting goals for the system which are then posted onto the blackboard. Each starting goal is a skeletal instance of the `Workgroup` class containing only those properties that describe its composition and constraints. There are no `hasMember` properties implying its membership. It is the posting of these goals onto the blackboard that kick-starts the whole process.

In the current AWB+B implementation this interaction is minimal, merely the problem definition. However, there is nothing to prevent a more "interactive" human KS. Another variation of a User KS could, for example, continually check the blackboard for inconsistencies and when one is found present the user with pop-up windows asking them to offer a possible resolution, i.e. it gives the user a "view" of inconsistencies found on the blackboard.

4.2 Instance based KS

This type of KS contains instance data corresponding to an ontology but not the actual schema itself. This could either be from a simple RDF file, a Web Service or data held in an RDF datastore. This KS contributes in the following way:

 i) Try to add a "solution" to a posted (sub-)goal by adding instance data for classes and/or properties defined on the blackboard.

ii) Try to add a "solution" to classify any property's *direct* subject and/or object which the blackboard does not have a class definition for.

For example, if the ontological class `Professor` is defined on the blackboard and this KS has instances of that class then, as per (i), the offered solution is all the `Professor` instances that it knows about. Property definitions work in the same way, but are slightly more complex. As per (i), when this KS responds to the property based goal `worksFor` the KS would offer the statement:

```
<ex:john>      <ont:worksFor>  <ex:abdnUni>          .
```

However, this gives no information about the subject or the object of that triple. This is not an issue if they are already instantiated on the blackboard, but if they are not (and assuming the object is not a literal) then subsequently, as per (ii), the KS could also offer the following:

```
<ex:john>      <rdf:type>      <ont:Lecturer>    .
<ex:abdnUni>  <tdf:type>      <ont:University>  .
```

Since this KS does not know the underlying schema, it cannot contribute class definition information about the `Lecturer` or `University` classes.

If this KS is a repository of RDF triples (e.g. 3Store [9]) then no reasoning ability is assumed. We require a wrapper for this KS, allowing us to communicate with the datastore via its API. In the case of the 3Store, it uses a `http` interface that accepts SPARQL[6] queries. We transform any blackboard goal into a query, the result of which can be transformed into triples and asserted onto the blackboard.

Since this type of repository can contain a vast amount of information, this raises the issue of the state which that information is in. Since access to the data is via a query mechanism, we are still effectively querying an RDF graph for which we have no means of knowing whether all, some or no additional entailments have been inferred. For example, while an ontology describes a `Professor` as a sub-class of `Academic` and the datastore contains instances of `Professor` for this schema, it might not actually contain the triples saying that `Professor` instances are also `Academic`s. Consequently, a SPARQL query for Academics would not return the Professors as it does not follow sub-class links. The only way around this is to query for all the sub-classes; which will eventually occur, as the Schema based KS (described next) will post the sub-classes as sub-goals prompting more refined queries.

4.3   Schema based KS

This represents a KS that only contains information at an ontological schema level. Since the blackboard initially contains no ontological structure (i.e. it does not contain any RDFS/OWL statements for the domain), it is the job of this KS to help facilitate the construction of the relevant ontological parts on the blackboard. This type of KS attempts to contribute in the following ways:

---

[6] SPARQL (SPARQL Protocol And RDF Query Language), is documented at: `http://www.w3.org/TR/rdf-sparql-query/`

i) Try to add new sub-goals to the blackboard by looking for:
   - Ontological sub-classes of a class defined on the blackboard.
   - Ontological sub-properties of property defined on the blackboard.
ii) Try to improve the (limited) reasoning ability of the blackboard by adding and inferring:
   - subClassOf statements connecting classes already defined on the blackboard.
   - subPropertyOf statements connecting properties already defined on the blackboard.

   Note: Statements are only added for *direct* sub-class/sub-property relations.
iii) Try to add new sub-goals for any property's subject and/or object on the blackboard that does not have a class definition. The sub-goals, in this case, being the missing class definitions.

In (i) and (ii) super-classes/properties are never added to the blackboard as these are deemed irrelevant and would widen the scope of the blackboard contents too much. Likewise, we need to be careful in (iii), as we do not just want to use the *domain* and *range* values of a property because they might (intentionally) be set very open. Continuing our previous example from section 4.3, let us suppose that when the ontology was first authored, the `worksFor` property was assigned a domain of `Person` and a range of `<owl:Thing>`. This was because the author believed that only a `Person` is capable of working, but what it is they actually work for could either be another *Person* or an *Organisation*. Therefore, for simplicity, they just widened the domain to encompass as many classes as possible. If we were to use these domain and range values, we would introduce a sub-goal asking for all instances of `<owl:Thing>` which would end up with each KS offering every instance it has. Therefore, in an attempt to narrow the search space as much as possible, only the class definitions of instances with the `worksFor` property are added as sub-goals (in this case `Lecturer` and `University` respectively).

4.4 Rule Based KS

A Rule KS, like all the other KS types, can be viewed as a black box, encapsulating its rules and keeping them private. The ability to derive new information through rules is an extremely important and powerful asset. Since we assume that these rules come from the open SW, we use SWRL[7] for expressing them as it is currently the dominant representation. This KS works by examining the contents of the blackboard to determine if any of the rules that it knows about are required and then attempts to contribute. A rule is required *only* if any of the elements in the consequent (head) are present on the blackboard[8]. The KS attempts to contribute to the blackboard in the following ways:

---

[7] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member submission, `http://www.w3.org/Submission/SWRL/`

[8] The reason why this is "any head element" is because SWRL allows the consequent to contain a conjunction of atoms.

i) Try to add a "solution" by firing the rule against instances already on the blackboard and asserting the appropriate triples.

ii) Try to add new sub-goals to the blackboard by looking for:
   - Any ontological classes that are antecedents of the rule and that have not been defined on the blackboard.
   - Any ontological properties that are antecedents of the rule and that have not been defined on the blackboard.

   The sub-goals in this case are the ontological class or property definitions.

We need to be careful here, remembering that we want to keep the blackboard contents relevant to the problem at hand and not introduce superfluous classes or properties. The following example illustrates one way a Rule Based KS can comply with this relevancy criteria of the blackboard:

"If a person is known to be an author of a book and that book has a topic then this implies that the person is an expert on that particular topic."

In informal SWRL syntax, where $?x$ denotes a variable, this can be written as:

(1) `Person`$(?p)$ $\wedge$ `Book`$(?b)$ $\wedge$ `hasTopic`$(?b,?t)$ $\wedge$ `authorOf`$(?p,?b)$
    $\Rightarrow$ `PublishedAuthor`$(?p)$ $\wedge$ `expertOn`$(?p,?t)$

Here `Person`, `Book` and `PublishedAuthor` are predicates denoting ontological classes and `hasTopic`, `authorOf` and `expertOn` are predicates relating properties to classes. If a rule has a conjunctive consequent it can be split into separate rules for each head atom. This means that if a consequent is not needed, that rule will not be considered avoiding the placement of unnecessary sub-goals (i.e. class/property definitions) onto the blackboard that could, subsequently, cause other KSs to add irrelevant information relating to these (either solution instances or sub-class/property sub-goals). Because of the conjunction in the consequent, this can be split up and re-written as 2 separate rules:

(2) `Person`$(?p)$ $\wedge$ `Book`$(?b)$ $\wedge$ `authorOf`$(?p,?b)$ $\Rightarrow$ `PublishedAuthor`$(?p)$
(3) `Person`$(?p)$ $\wedge$ `Book`$(?b)$ $\wedge$ `hasTopic`$(?b,?t)$ $\wedge$ `authorOf`$(?p,?b)$
    $\Rightarrow$ `expertOn`$(?p,?t)$

Using rule (1) from the above example, if the blackboard only contains the property `expertOn` then we apply the equivalent of rule (3) to the blackboard and since (2) is not needed, it will not be considered – we only want to add new `expertOn` properties and ignore instances of the class `PublishedAuthor`.

Now if `PublishedAuthor` does exist on the blackboard and we assume no new antecedent information has been placed on the blackboard between cycles, the first cycle would involve adding all values of `PublishedAuthor` as per rule (2). In the 2nd cycle, it would see that no new instance data for `PublishedAuthor` can be added and, iteratively, move onto the next consequent item (`expertOn`) to see if it can contribute as per rule (3) – which it can. Finally, in the 3rd cycle, the rule would check both (2) and (3) and find that no new data can be added, so it would respond by saying it cannot contribute.

The current implementation of the AWB+B rewrites each rule into SPARQL queries, which it places against the blackboard contents to determine if any

new triples can be asserted – essentially this is a *brute force*, forward chaining approach to deriving new entailments. The query results are then added to the blackboard. From the previous example, using rule (3) to determine the `expertOn` property, we can create the following query, where `rdf` is the RDF namespace, and `ont` is the appropriate ontology namespace:

```
SELECT ?p, ?t
WHERE { ?p  rdf:type     ont:Person .
        ?b  rdf:type     ont:Book   .
        ?b  ont:hasTopic ?t         . }
```

The result set of this query contains value pairs of *Person* URI and *Topic* URI which are asserted onto the blackboard as the new triples of the form:

```
?p  <ont:expertOn>  ?t .
```

We avoid adding duplicate instance values by checking the query results and only adding new values. Since multiple rules could be applied, with new antecedent instances added at any time by any of the KSs, this is an iterative process with the entire sequence repeated until no new entailments are generated.

While this may not be the most efficient implementation, it does reflect the opportunistic nature of a blackboard system. The current implementation only has one starting rule per Rule KS (it can be split as per the above example), but we plan to extend this to allow multiple, different rules within one KS. This also allows for the interdependency between each of the rules within a KS as well as rule chaining.

From the Blackboard's perspective, all the KSs are goal oriented (backward chaining). When a rule based KS posts a sub-goal it is understood that it (the KS) has concluded that, by backward chaining, a solution can be posted once this sub-goal has been achieved. However, internally it could be forward chaining. Suppose the KS in question is an efficient implementation of a RETE forward chaining algorithm. The KS works by constantly monitoring the blackboard's contents and, within its own Knowledge Base, duplicates all elements it requires to forward chain using its known rules. Once all the processing has completed, it then offers those newly derived facts as solutions to goals posted on the blackboard. In this scenario the KS might never post a single sub-goal onto the blackboard, merely offering solutions to blackboard goals.

In our future work we wish to further investigate the trade-off between opportunism and relevancy of the blackboard data, specifically by examining the effect that *Rule Chaining*, with its inter-dependent antecedents, will have on the posting of sub-goals onto the blackboard. If a KS encapsulates its rule's antecedents then these will never be placed upon the blackboard as sub-goals. Our main concern is how this will effect the reasoning process. Since the other KSs are not aware of these potential sub-goals (which they may have instances of, and hence solutions too) they will never make a relevant contribution. Therefore, the overall potential solving ability of the system decreases, due to this unnecessary curtailment of the solution space. Conversely, it might be the case that these intermediate antecedents are not actually required/relevant at all and so by placing them as sub-goals on the blackboard a whole raft of irrelevant in-

stance data (and consequential sub-goals) would placed upon the blackboard. This would be detrimental to the solving process as effort could be expended on non-relevant data.

4.5 CSP Solver KS

This KS allows us to perform constraint-based reasoning and attempts to solve the CSP goal posted on the blackboard. The constraints for the workgroup(s) are expressed using CIF/SWRL [13] – our Constraint Interchange Format (CIF), which is an RDF based extension of SWRL that allows us to express fully quantified constraints. These constraints are placed on the blackboard by the User KS when the workgroup is first defined. Since the goal of the AWB+B is to form workgroups that adhere to these specified constraints, this KS has the trigger:

i) Try to add a "solution" by using instance data already on the blackboard to perform CSP solving and assert the appropriate `hasMember` triples to the corresponding instance of the `Workgroup` class.

The triggering mechanism of this KS requires it to continually monitor the blackboard contents and attempt to provide a solution to the CSP. To improve efficiency, we decided that rather than attempting full blown CSP solving each cycle, the solver should perform the faster check of each of the constraints individually and only if they can all be satisfied, should it attempt the more difficult task of solving them combinatorially. If no solution can be found then this KS will simply not offer a contribution.

In our implementation the CSP solver is unique, in that it is the only KS that can post a solution to the `Workgroup` goal, initially posted onto the blackboard by the User KS. However, there is no restriction on the number of CSP solver KSs that could be used within the system. In our future work there is also the possibility of greater user interaction (via the User KS) w.r.t to acceptance or rejection of a solution. Here the user could ask the CSP Solver KS to contribute again (provided there are alternate solutions) or accept the current one.

5 The Controller

As the name suggests, the role of the Controller is to oversee the running of the system as a whole. In the initial blackboard systems, one of the main problems was the lack of direction or a statement of goals to focus the solving effort. The BB1 system [10] extended the blackboard architecture by adding a second blackboard to control the state of the problem, and so better direct the solving. So far we have talked about the contents of the blackboard as merely containing the solution. In actual fact, like BB1, the AWB+B blackboard is divided into two panels. The first panel is the Data Panel which holds the solution related information. In order to inhibit the actions of the KSs accessing this panel, there are a couple of safeguards in place. The Controller will not allow the goal of "instances of `<owl:Thing>`" to be placed onto the blackboard since this is *the*

OWL super-class, this would result in *all* class (and sub-class) instances known by a KS being added onto the blackboard.

The second panel is the Tasklist Panel and is used, primarily, by the Controller to coordinate the actions of each KS by storing information about *what* each knowledge source can contribute, based on the current state of the blackboard. Like the Data Panel, this is visible to all the KSs however, unlike the Data Panel, the KSs are allowed to add to this panel directly (but not remove items from it). The KSs add `TasklistItems` that describe the nature of any contribution they could offer. The Controller looks at the items on the Tasklist Panel and determines which KS is allowed to contribute. Once a `TasklistItem` has been actioned, the Controller removes it from the panel. This "request for contribution" and "make your contribution" sequence is applied using a Java interface, which each registered KS must implement and consists of the two method calls: `canContribute` and `makeContribution`. When a KS's `canContribute` method is called it first determines *what* it can contribute (as per the steps previously outlined) and then checks, in the following order, that its "current" proposed contribution is not on the blackboard already; has not been contributed previously by itself; and is not already on the Tasklist, i.e. already proposed by another KS. Only then is a `TasklistItem` created and added to the Tasklist Panel.

In our current implementation the Controller is relatively simple. After all the KSs have been registered, the system "cycles" over each one asking it to populate the Tasklist Panel. Next, the Controller examines the contents of the Tasklist and decides which items to action (by calling the appropriate `makeContribution` method of a KS). After actioning the appropriate `TasklistItems` on the Tasklist Panel, the Controller has the option of retaining tasks that have not been actioned, or removing any remaining items from the Tasklist completely. This is purely a housekeeping measure as it prevents redundant or "out of date" items remaining on the Tasklist Panel. Then the cycle begins again. If nothing new has been added to the Tasklist Panel after a complete cycle, it is assumed that none of the KSs can contribute further and so the CSP Solver KS is activated and attempts to find a solution. While this is relatively straightforward to implement, it is far from optimised. We plan to increase the intelligence of the Controller to further focus the problem solving, which should improve performance.

6 Future Direction

Currently, the CSP solver adopts a closed world reasoning model. Since the Semantic Web is open world, it seems logical to investigate the influencing and enabling factors for performing effective open world reasoning based upon closed world reasoners as well as the analysis and combination of constraints to solve the CSP. The importance of negation and negative information is argued by Analyti and Wagner in [1]. The difficulty here is that negation is only partially supported in OWL DL (via disjunction) but not at all in OWL Lite. This would require either the extension of OWL at the language level in much the same way

as the RDF(s) extension suggested in [1]; or through the application of rules to explicitly state negative information (as described in [14]).

We plan to incorporate the *Local Closed World Assumption* (LCWA) [7, 4] into the AWB+B, since this can be thought of as a compromise between both the open and closed world assumptions. The LCWA means that a query placed against a Knowledge Base (KB) can be answered with *true*, *false* or *unknown*. For example, if we posed the query "Is *Person(x)* also a *Professor*?" against our KB, what answer would we get? With both assumptions, if the result *Professor(x)* or ¬*Professor(x)* exists in the knowledge base (either explicitly or via some derived means) then the result would be *true* or *false* as appropriate. However, if there is no explicit statement describing x as a *Professor* then in the open world assumption, the result would be *unknown* since we cannot prove this definitively either way. In the case of a closed world the result would be *false* – the assumption being made that if we do not know a fact for definite, then it is always regarded as *false*. Now, suppose we do know that there is only one *Professor* in the world – *Professor(y)* and, therefore it is not *Person(x)*. How could we guarantee that our query correctly returns *false*? With the closed world assumption this would already be the case, however with the open world assumption we would still have the value *unknown*. The only way to correct this error would be to state the fact ¬*Professor(?)* for the (potentially infinite) set of everything else in the world. The LCWA overcomes this problem by maintaining two databases of world information. The first contains the known facts describing the world. The second one contains metadata indicating what sets of facts in the first database can be regarded as closed (since it would be impossible to store a potentially infinite set of non-members). So, using our example above, the first database would contain the fact *Professor(y)* and the second one would contain *ClosedWorld(Professor)*. Using the query mechanism of the LCWA we could extend how data is contained within each KS and on the blackboard, whereby the result of a query is either *true*, *false* or *unknown*. This allows the option of creating a specific goal as an attempt at resolving an *unknown* that would otherwise have been regarded as a *false*.

This process should stop when we believe that the closed world problem is now complete enough to enable realistic problem solving. A closed collection of values is crucial since the problem solving we are attempting is that of a Finite Domain CSP. When a generated workgroup is "released" back onto the open Semantic Web, the composition will need some explanation as to the assumptions used in its creation (i.e. it has been created using negation as failure). Therefore we require an appropriate representation that would better enable someone else to reuse the data and to reason against further. For example, if a published workgroup is constrained to only contain *Student*s but, on closer examination, one member is actually a *Lecturer* then this is a contradiction. By annotating the composition such that it is clear that it was constructed using a closed world, negation as failure, approach, a consumer may be more forgiving as their assumption would be the *Lecturer* was simply misclassified at the time of construction due to a lack of data.

7 Discussion and Conclusion

In this paper we have explained why we believe the blackboard architecture aligns itself well with the hybrid reasoning approach necessary for reasoning with Semantic Web data. An important issue we encountered was the inefficiency of the two stage "trigger" and "action" approach to KS interaction with the blackboard. The work involved for a KS to know if it is "triggerable" is comparable to that of making the actual contribution itself, which it may never be called upon by the controller to do. However, we believe the benefits of the blackboard architecture outweigh this shortcoming since the paradigm allows us to perform a mix of reasoning methods on instance data and we would argue that performance could be improved by committing more time to the development of the AWB+B to further optimise the triggering conditions.

There are also complexity issues to be considered with the Blackboard framework giving us a number of options on how to explore these in our future work. For example, what if a KS starts to perform reasoning that could take hours or days to complete? The architecture supports the addition or removal of KSs from the system with the only adverse effect being on quality of the results and so guards against the inefficiency of KSs – the overall process of controlling the problem solving remains with the Controller. For example, had we implemented an asynchronous version of the application, then a time-out mechanism could be added to the Controller, so if a KS takes an inordinate amount of time to respond it could just be ignored, allowing the rest of the system to continue.

We have also highlighted the importance of ensuring only *relevant* items are placed on the blackboard and how this effects the opportunism of the system. Since the blackboard system is attempting to centralise distributed SW data it does not want *all* the data available from each of the KSs; it is only interested in as small a subset of this as is possible in order to solve the CSP problem. It is the job of the Controller to ensure that this is the case, otherwise it may become intractable. Since we place a great deal of importance upon relevancy, the high level strategy of the controller is that of a goal driven (backward chaining) approach – in our case, the initial goal placed on the blackboard is to find a solution to a specified workgroup problem. In our future work there is the possibility of adopting a forward chaining approach, overseen by a suitable controlling strategy. We also believe there is scope for further investigation of complexity and scalability trade-offs (e.g. using multiple ontologies that require mapping, increasing the size of the dataset, etc) as well as the modification of the Controller strategy to enhance performance.

References

1. C. V. D. Anastasia Analyti, Grigoris Antoniou and G. Wagner. Negation and Negative Information in the W3C Resource Description Framework. *Annals of*

    Mathematics, Computing & Teleinfomatics, 1(2):25–34, 2004.

2. R. Brachman, V. Gilbert, and H. Levesque. An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON. In *The Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 532–539, Los Angeles, California, USA, 1985.

3. N. Carver and V. Lesser. The Evolution of Blackboard Control Architectures. CMPSCI Technical Report 92-71, Computer Science Department, Southern Illinois University, 1992.

4. P. Doherty, W. Lukaszewicz, and A. Szalas. Efficient Reasoning using the Local Closed-World Assumption. In *9th International Conference on Artificial Intelligence: Mehtodology, Systems, Applications*, 2000.

5. R. S. Engelmore and A. Terry. Structure and Function of the CRYSALIS System. In *The Sixth International Joint Conference on Artificial Intelligence (IJCAI79)*, pages 250–256, Tokyo, Japan, August 20-23 1979.

6. L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *ACM Computing Surveys*, 12(2):213–253, 1980.

7. O. Etzioni, K. Golden, and D. Weld. Sound and Efficient Closed-World Reasoning for Planning. *Artificial Intelligence*, 89(1-2):113–148, January 1997.

8. E. A. Feigenbaum, H. P. Nii, J. J. Anton, and A. J. Rockmore. Signal-to-signal Transformation: HASP/SIAP Case Study. *AI Magazine*, 3(2):23–35, 1982.

9. S. Harris and N. Gibbins. 3store: Efficient Bulk RDF Storage. In *1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pages 1–20, 2003.

10. B. Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence*, 26(3):251–321, 1985.

11. B. Hayes-Roth, F. Hayes-Roth, F. Rosenschien, and S. Cammarata. Modelling Planning as an Incremental, Opportunistic Process. In *The Sixth International Joint Conference on Artificial Intelligence (IJCAI79)*, pages 375–383, Tokyo, Japan, August 20-23 1979.

12. J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30–37, March/April 2001.

13. C. McKenzie, A. Preece, and P. Gray. Extending SWRL to Express Fully-Quantified Constraints. In G. Antoniou and H. Boley, editors, *Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, LNCS 3323, pages 139–154, Hiroshima, Japan, November 2004. Springer.

14. J. Mei, S. Liu, A. Yue, and Z. Lin. An Extension to OWL with General Rules. In G. Antoniou and H. Boley, editors, *Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, LNCS 3323, pages 155–169, Hiroshima, Japan, November 2004. Springer.

15. H. P. Nii. Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Magazine*, 7(2):38–53, 1986.

16. N. Shadbolt, N. Gibbins, H. Glaser, S. Harris, and m. schraefel. CS AKTive Space, or How We Learned to Stop Worrying and Love the Semantic Web. *IEEE Intelligent Systems*, 19(3):41–47, 2004.

17. D. Tsarkov and I. Horrocks. DL Reasoner vs. First-Order Prover. In *2003 Description Logic Workshop (DL 2003)*, volume 81, pages 152–159. CEUR (http://ceur-ws.org/), 2003.