A Semantic Web 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 propose a Blackboard Architecture as a means for coordinating hybrid reasoning over the Semantic Web. We describe the components of traditional blackboard systems (Knowledge Sources, Blackboard, Controller) and then explain how we have enhanced these by incorporating some of the principles of the Semantic Web to produce our Semantic Web Blackboard. Much of the framework is already in place to facilitate our research: the communication protocol (HTTP); the data representation medium (RDF); a rich expressive description language (OWL); and a method of writing rules (SWRL). We further enhance this by adding our own constraint based formalism (CIF/SWRL) into the mix. We provide an example walk-though of our test-bed system, the AKTive Workgroup Builder and Blackboard(AWB+B), illustrating the interaction and cooperation of the Knowledge Sources and providing some context as to how the solution is achieved. We conclude with the strengths and weaknesses of the architecture.

1 Introduction & Motivation

Since the Semantic Web (SW) is essentially a symbolic version of the current Web, anyone attempting to use published data is still faced with handling the standard knowledge integration and reuse problems [12, 3]: accommodating errors in the data at either a syntactic or semantic level, incompleteness, inconsistency, intractability, etc. Performing reasoning of various kinds in order to "do the best we can with what we've got" is a hard problem. The primary goal of our research is to investigate the suitability of a Blackboard System as a means of co-ordinating hybrid reasoning over the SW.

The SW is intrinsically hybrid in terms of its knowledge representation formalisms. Our ground data exists as RDF which will normally conform to either an RDF Schema or an OWL ontology[1]. The semantics of these mean that we are able to perform different types of reasoning upon them – ranging from transitive closure right up to full DL (ABox and/or TBox[7]) classification – to better enrich the data by deducing facts that may not have been explicitly stated. Derivation rules, represented using SWRL[2], can also be applied in order to generate additional entailments.

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

We believe that reasoning on the SW requires a combination of reasoning methods rather than just a single "super-reasoner". For example, [16] compared a DL reasoner to a first-order theorem prover, and conclused 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 identified slow performance speed as a potential hurdle. Advocacy of a hybrid approach to reasoning predates the SW [1]. However this is not without its problems. How should contradictions be handled? Can conflicting reasoning strategies interfere with one another? Hence, some mechanism is required to help manage such issues. However there is currently nothing in the SW architecture for coordinating this effort, so we believe that 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 [13, 2].

Having outlined our reasons for our research, in the following section (Section 2), we introduce the problem domain and our test-bed Blackboard System. In Section 2 we discuss the blackboard architecture and explain the role of each of its constituent parts, then in Section 3 we discuss the changes to this in our Semantic Web approach. In Section 5, we perform a walk-through of the application to illustrate the interplay between the component. Section 6 is a discussion of how we focus the reasoning effort. In Section 7 we describe the issues we encountered and outline our future work before discussing our final conclusions in Section 8.

2 Problem Domain and the AWB+B Application

We needed to decide the problem domain within which to we wished to work. We settled upon the same context as that of the CS AKTive Space [15], namely the Computing Science (CS) community in the UK. The data describes people, their research interests, affiliations, publications and projects at various levels of granularity and completeness.

Our demo application, called the AKTive Workgroup Builder and Blackboard (AWB+B)[3], is a web-based application that utilises disparate RDF based information in constructing a workshop, containing one or more working groups of people, from this pool of known individuals. Each workgroup must adhere to a set of user defined constraints, e.g. "the workgroup must contain between 5 and 11 individuals" or "at least half the members of the workgroup must be students". 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 and accessible via URIs.

The final aspects of the problem were the representation of constraints. For this, we had already developed an ontology as a means of representing Constraint

[3] This is a refinement of the non-blackboard AWB system in [11].

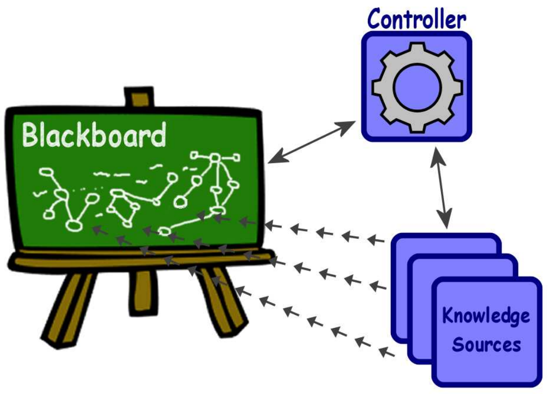

Fig. 1. The core architectural components of a Blackboard System. Each KS can view the contents of the Blackboard but it is the Controller that decides which KS(s) are allowed to contribute.

Satisfaction Problems (CSPs) on the SW[14]. This also builds upon our earlier work, CIF/SWRL [11], in developing a SW representation for fully-quantified, individual constraints. This is based on SWRL and is used for expressing the individual constraints that comprise a CSP.

3 Blackboard Architecture

Back in the late '70s, when Lesser and Erman developed Hearsay-II [10] they conceived the Blackboard Architecture [6, 9] as an effective means for collaborative problem solving. The premise was simple: *how would a group of real-life experts work together to solve a complex problem?* Since none of the individuals were capable of solving it on their own, they would all gather around a blackboard, with each person using their knowledge and expertise to contribute by either: decomposing a problem into smaller sub-problems and writing these on the board; or solving an existing problem and writing up the answer. Slowly, the overall problem would be progressed until, finally, it reached a state where it was solved (for a fuller description see [13]). The key aspects of this approach being that the solving process is only possible through collaboration (no individual is capable of solving the problem on their own) and in an incremental manner (progress is made via small steps).

The whole process is overseen by a *Controller* that performs two roles. Firstly, it enforces a protocol for who gets to write on the blackboard and when. Returning to the metaphor, if there is only one piece of chalk, the controller would decide whom gets to use it and when. If there is a whole box of chalk, the controller would ensure that all the writers do not get in one another's way. Secondly, the controller attempts to keep the contents of the blackboard relevant by asking each KS what sort of contribution they can make before deciding to let them make it or not (see Figure 1).

As stated by Wooldridge in [17], “Blackboard Systems were recognisably the earliest form of a Multi-Agent Systems”. Now, over a quarter of a century later, we believe that the blackboard paradigm is still a valid one (in fact, recent work has focussed on [4, 5] their suitability for collaborating software), especially when used in concert with the newer technologies that make up the SW. The following subsections describe the roles of the three main components of the traditional Blackboard architecture (the Knowledge Sources, the Blackboard itself and the Controller) before discussing in more detail our SW based approach.

3.1 The Knowledge Sources (KSs)

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. In implementation terms, KSs do not interact with one another, nor even know about any of the others that are present. Also, no assumptions should be made about the capabilities of a KS – conceptually it should be regarded as a black box. Each KS has a precondition, or event trigger, (indicating that it can add something to the blackboard) and an action (what it can add to the blackboard). Due to the tightly coupled nature of the KSs and the Blackboard, all KSs must be “registered” so that they can continually check the blackboard and determine if they can make a contribution. 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).

3.2 Knowledge on the Blackboard

The actual blackboard itself can be thought of as a shared data repository representing a communal work area or “solution space” of potential solution components. Since the KSs are required to both view and modify the contents of the blackboard it is also a communication medium. For all this to work efficiently, the data held on the blackboard is structured hierarchically into what were called *abstraction Levels*. If the blackboard contained multiple distinct hierarchies, these were referred to as *panels*.

This organisation served two purposes. Firstly, it aided each KS in checking 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 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.

3.3 The Controller

As mentioned already, the decision of what is (or is not) placed on the blackboard is made by the controller. The complexity of this strategy can vary from a simplistic "just action everything" approach to a more complex goal driven algorithm. The key point is that the controller 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 and also means that 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. It should also be noted, that the current implementation of the AWB+B, the blackboard is monotonic, facts are only ever added by the KSs, never retracted.

4 The Semantic Web Approach

Our Semantic Web Blackboard maintains all the principles of a traditional blackboard but improves upon it by incorporating some of the concepts of the SW. The notion of abstraction levels aligns itself well to the hierarchical, structured nature of an ontology. Historically, abstraction levels were developed at design time so their structure was fixed throughout the execution of the system. In the AWB+B, the information represented on the blackboard is stored as a dynamically created RDF graph. RDF statements ("triples") can be added incrementally to the blackboard to gradually build up both ontological information and instance data. For example, we can add the triples

```
  <ex:Tim>  <rdf:type>  <ont:Lecturer>
  <ont:Lecturer>  <rdfs:subClassOf>  <ont:Academic>
```

in any order, resulting in the knowledge that some instance Tim is a lecturer (instance-level data) and that a lecturer is a kind of academic (ontological data).

To the best of our knowledge, in the past the blackboard has always been passive with any deductive mechanism performed by 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 ontological structure evolving on it. Four rules are forward chained (here, clauses are RDF triples, and `?x` denotes a variable):

i) `(?a <rdfs:subClassOf> ?b)` & `(?b <rdfs:subClassOf> ?c)`
$\Rightarrow$ `(?a <rdfs:subClassOf> ?c)`

```
 ii) (?x <rdfs:subClassOf> ?y) & (?a <rdf:type> ?x)  ⇒  (?a <rdf:type> ?y)
iii) (?a <rdfs:subPropertyOf> ?b) & (?b <rdfs:subPropertyOf> ?c)
                                        ⇒  (?a <rdfs:subPropertyOf> ?c)
 iv) (?a ?p ?b) & (?p <rdfs:subPropertyOf> ?q)  ⇒  (?a ?q ?b)
```

Here we are only materialising all the transitive sub-class/property relations and all the instance type relations. For example, as per rule (i), 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 (rule (ii)). 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. Rules (iii) and (iv) are similarly applied for properties.

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 must be treated with caution. It would be undesirable if the blackboard became a bottleneck while it attempted to fully reason about facts posted upon it while denying all the KSs from contributing (especially if a "denied" KS was attempting to add a fact that would help the reasoning process). This is why we have not increased the blackboard's inference ability any further.

5 A Walk-though of Cooperative Problem Solving

Having now described the components of the blackboard architecture, we now use a couple of the possible KS types in order to present a walk-through of a simplified problem – constructing a single workgroup. This should illustrate the cooperative and incremental nature of a blackboard system. Since the contents of the blackboard in the AWB+B is an RDF graph, we have used a simplified form of RDF to illustrate this throughout.

5.1 Human (User Interface) KS

While not immediately obvious, the user of the system can be regarded as a type of KS representing the "human knowledge" of the system in the form of "human input". In the AWB+B case, this is the user entering the initial system parameters (via a web interface), i.e. the number of workgroups to be built, the size of each workgroup, any associated compositional constraints, derivation rules etc. This information is then transformed into the system's starting goals and posted onto the blackboard.

In the current AWB+B implementation human interaction is limited to that outlined above. 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.

Now, for our running example, let us suppose the user wishes to compose a single workgroup, as per the following constraints:

1. Must contain between 3 and 5 members, of type `Person`.
2. Must contain at least 1 `Professor`.
3. Must contain an `expertOn` "Machine Learning".

In addition to this, the user also specifies a SWRL rule, paraphrased as: "if a person is an author of a book and the subject of that book is known, then this implies that the author is an expert on that subject". This is written in SWRL as follows:

```
Person(?p) & authorOf(?p, ?b) & Book(?b) & hasSubject(?b, ?s)
                                          ⇒  expertOn(?p, ?s).
```

All this results in the following skeletal workgroup structure being placed on the blackboard, as well as the class definitions for `Person` and `Professor`, and a property definition for `expertOn` (from the specified constraints):

```
<ex:Wg1>  <rdf:type>  <wg:Workgroup>
<ex:Wg1>  <wg:hasMinMembers>  3
<ex:Wg1>  <wg:hasMaxMembers>  5
<ex:Wg1>  <wg:hasFillerClass>  <ont:Person>
<ex:Wg1>  <wg:hasConstraint>  <ex:OneProfessor>
<ex:Wg1>  <wg:hasConstraint>  <ex:MLExpert>
<ont:Person>  <rdf:type>  <owl:Class>
<ont:Professor>  <rdf:type>  <owl:Class>
<ont:expertOn>  <rdf:type>  <rdf:Property>
```

(Here, `<ex:OneProfessor>` and `<ex:MLExpert>` are references (URIs) to the constraints (2) and (3) above, which are expressed in our CIF/SWRL language.

5.2 Instance-based KS

This type of KS contains only 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. We cannot assume that any additional entailments have been generated for the RDF as this KS may or may not have a reasoner attached to it. 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.

If this KS is a repository of RDF triples (e.g. 3Store [8]) we require a wrapper for it, allowing us to communicate with the datastore via its API. In the case of the

3Store, it has an HTTP interface that accepts SPARQL queries[4]. 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 `Academics`. 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. However this will eventually occur because the Schema based KS (described next) will post the sub-classes as sub-goals which will prompt more refined queries.

Continuing our example, we have three goals on the blackboard (two classes defined, namely `Person` and `Professor`, and a property definition, `expertOn`). If this KS has instances of `Professor`, then it will offer these as solutions (as per (i)). Property definitions work in the same way, but are slightly more complex. Let us assume, our KS has a statement relating to the `expertOn` property, and therefore offers this statement:

```
  <ex:Tim>  <ont:expertOn>  "Semantic Web"
```

However, this gives no information about the subject, `<ex:Tim>`, of that triple (the same would have applied for the object, `"Semantic Web"`, had it not been a literal). This would not be an issue if `<ex:Tim>` was already instantiated on the blackboard, but since it it not, then this KS will subsequently offer (as per (ii)):

```
  <ex:Tim>  <rdf:type>  <ont:Lecturer>
```

Because this KS does not know the underlying schema, it cannot contribute class definition information about the `Lecturer` (e.g. what this might be a sub-class of).

5.3 Schema-based KS

This represents a KS that contains only ontological schema information. Since the blackboard initially contains no ontological structure other than the starting goals, 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:

i) Attempt to add new sub-goals by looking for ontological sub-classes/sub-properties of those already defined on the blackboard.
ii) Attempt to improve the (limited) reasoning ability of the blackboard by adding `<rdfs:subClassOf>` or `<rdfs:subPropertyOf>` statements connect-

[4] http://www.w3.org/TR/rdf-sparql-query/

ing those already defined on the blackboard. These connective statements are only added for *direct* sub-class/sub-property relations.

iii) Attempt to add new sub-goals for any subject/object on the blackboard that does not have a class definition. The sub-goals, in this case, would be the missing class/property 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 want to simply use a property definition and add the classes specified in its domain and range values as new sub-goals. This is because they could introduce non-relevant goals onto the blackboard, instead, we just use the classes of actual instances that have this property. To clarify, let us suppose that when the ontology was first authored, the `expertOn` 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 being an expert, but what it is they have expertise in could be anything. 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 result in each KS offering every class instance it knows about. Therefore, in an attempt to narrow the search space as much as possible, only the class definitions of instances with the `expertOn` property are added as sub-goals.

Continuing our running example, based upon the current contents of the blackboard, this KS would see the class definition of `Person` and act upon it by offering to add a sub-goal, as per (i), by defining a sub-class of Person:

```
<ont:Academic> <rdf:type> <owl:Class>
```

In blackboard terms, defining the class `Academic` does not automatically specify it as a sub-class of `Person`. This must explicitly be stated on the blackboard. Therefore, this KS would next offer the explicit sub-class link between these two classes (as per (ii)):

```
<ont:Academic> <rdfs:subClassOf> <ont:Person>
```

Finally, this KS would see the statement (previously contributed by the Instance KS):

```
<ex:Tim> <rdf:type> <ont:Lecturer>
```

The triple already implies that `Lecturer` is a class (although, it could either be a `rdfs:Class` or the more specific `owl:Class`). The KS would then offer the full class definition (as per (iii)), hence re-classifying Tim appropriately and creating a new sub-goal, so from our earlier examples we would now have:

```
<ont:Lecturer> <rdf:type> <owl:Class>
<ex:Tim> <rdf:type> <ont:Lecturer>
<ex:Tim> <ont:expertOn> "Semantic Web"
```

5.4 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. We achieve this by expressing them using SWRL, although there is no restriction on what rule representation is used (especially since the rules within a KS are private) we elected to use SWRL because it is part of the SW framework.

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[5]. The KS attempts to contribute to the blackboard in the following ways:

i) Try to add a “solution” by firing the rule against instances already on the blackboard and asserting the appropriate statement(s).
ii) Try to add new sub-goals to the blackboard by offering class/property definitions that are antecedents of the rule and have not been defined on the blackboard.

Reusing our derivation rule to determine expertise (Section 5.1), and continuing our example, we see that the Blackboard contains a class definition for `Person` but no property definitions (or instances of) the other rule antecedents, i.e. the class `Book` and the properties `authorOf` and `hasSubject`. Therefore, regardless of instance data, the rule is incapable of firing. Hence, this KS would offer the following sub-goals, as per (ii):

```
<ont:Book> <rdf:type> <owl:Class>
<ont:authorOf> <rdf:type> <rdf:Property>
<ont:hasSubject> <rdf:type> <rdf:Property>
```

Once other KSs have contributed instance data for the antecedents, the rule can fire and generate a solution instance for the `expertOn` property (i.e. backward chaining) that has not been explicitly stated in a KS (as per (i)).

5.5 CSP-solving KS

The final component of the AWB+B system is the CSP solving. The constraints for the workgroup(s) are expressed using CIF/SWRL [11] – 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 Human KS when the workgroup is first defined. Since the goal of the AWB+B is to form workgroups that adhere to these specified constraints, a CSP KS was created, having 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

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

cycle, the solver should perform a 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[6]. 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 on the blackboard.

6 Controlling Content

So far we have talked about the contents of the blackboard as merely containing data relating to finding a workgroup solution. In actual fact, 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 `<owl:Thing>` to be placed onto the blackboard and KS access to the blackboard is via a restrictive API that allows the underlying graph to be viewed by the KSs while not allowing it to be modified without the controller's knowledge.

The second panel is the Tasklist Panel, and is used by the controller to coordinate the actions of each KS by storing information about *what* each KS 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 purpose of this is to facilitate the controller in directing the solving effort. 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 in the KS descriptions) and then checks, in the following order, if its "current" proposed contribution is not

[6] It is possible that an instance-based KS could contain instances of workgroups and offers to contribute those. In our case, because the user specifies the KSs at the start, we assume that none of the KSs contain workgroup instances. Similarly, a workgroup could be formulated based on a rule set within a rule-based KS and offered as a solution.

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 if none of these cases apply is a `TasklistItem` created by the KS 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 (by calling its `canContribute` method). 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 after a complete cycle, it is assumed that none of the KSs can contribute further and the CSP Solver KS is activated and attempts to find a solution.

7 Issues & Future Work

One issue with the blackboard architecture is that the two step `canContribute` and `makeContribution` process is inefficient. The effort involved to determine whether a contribution can be made is comparable to actually making the contribution itself and even then, depending upon the controller strategy, the contribution my never be asked for. This overhead may be reduced somewhat by caching the result of the `canContribute` step so that if the KS is asked to make its contribution a duplication of effort is not required.

Another important issue is that of contradictory information being placed on the blackboard. In the current implementation of the AWB+B, contradictions are just ignored – they remain unresolved with the blackboard containing both discrepant parts. Should one of these be parts be required in the composition of a solution then it is just used. One possibility to improve upon this is to have a KS continually checking the blackboard, prompting the user should an inconsistency be found. This would enable the user to decide which fact they wish to retain and remove the remaining inappropriate data. At first glance this might appear to be a very appropriate course of action, since a human should be able to do a better job of deciding than a machine. Unfortunately, in the case where a large quantity of contradictions occur this becomes far from ideal, especially from a usability viewpoint since this would very quickly become unworkable. The user would be constantly attending to these notifications, making themselves a bottleneck and impacting the overall performance of the system (not to mention becoming increasingly frustrated). Therefore, a more automated approach is desirable and is an area in which we plan to investigate further.

We have also highlighted the importance of ensuring only relevant items are placed on the blackboard. Since the blackboard system is attempting to centralise distributed SW data it does not want all the available data 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. Since it is the job of the controller to ensure that this is the case, in our future work we plan to investigate possible controller strategies to improve relevancy, and therefore enhance the system performance.

8 Conclusions

Since reasoning is hard, we deem that reasoning over a dynamically composed sub-set of all the available data is preferable than just combining all that data *en masse* and processing that. To create as small a sub-set as possible it is important that the collected data is as relevant as possible to the problem at hand. We believe that the Blackboard Architecture is a suitable paradigm for controlling this effort since it not only enables a mix of reasoning methods but also allows them to operate cooperatively. This paradigm also supports the addition and removal of KSs from the process, even during runtime. For example, consider the scenario of a KS that starts to perform reasoning that could take hours, or even days to complete. Normally a system would have to wait until this was resolved before continuing. The Blackboard Architecture guards against the inefficiency of KSs (caused by numerous factors, e.g. tractability, network connections, etc) – the overall process of controlling the problem solving remains with the controller. Had we implemented an asynchronous version of the application, then a time-out mechanism can be added, so if a KS takes an inordinate amount of time to respond it is just ignored. The only adverse effect being on quality of the results.

Our SW Blackboard system is domain independent. Since the content of the blackboard is a dynamically generated RDF graph, a by-product of this is that it contains what is essentially a new *sub-ontology* representing the *relevant* parts of the problem domain too. This may be useful for a system geared more toward information gathering.

Acknowledgements. This work is supported under the Advanced Knowledge Technologies (AKT) IRC (EPSRC grant no. GR/N15764/01) comprising Aberdeen, Edinburgh, Sheffield, Southampton and the Open Universities. http://www.aktors.org

References

1. 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.
2. N. Carver and V. Lesser. The Evolution of Blackboard Control Architectures. CMPSCI Technical Report 92-71, Computer Science Department, Southern Illinois University, 1992.
3. C. Chweh. Generations ahead: Michael Huhns on cooperative information systems. *IEEE Intelligent Systems*, 12(5):82–84, September/October 1997.

4. D. D. Corkill. Collaborating Software: Blackboard and Multi-Agent Systems & the Future. In *Proceedings of the International Lisp Conference*, New York, New York, October 2003.
5. D. D. Corkill. Representation and Contribution-Integration Challenges in Collaborative Situation Assessment. In *Proceedings of the Eighth International Conference on Information Fusion (Fusion 2005)*, Philadelphia, Pennsylvania, July 2005.
6. R. S. Engelmore and A. J. Morgan, editors. *Blackboard Systems*. Addison-Wesley, 1988.
7. G. D. Giacomo and M. Lenzerini. Tbox and Abox Reasoning in Expressive Description Logics. In *KR-96*, pages 316–327, Los Altos, 1996. M. Kaufmann.
8. 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.
9. V. Jagannathan, R. Dodhiawala, and L. Baum, editors. *Blackboard Architectures and Applications*. Academic Press, 1989.
10. V. R. Lesser and L. Erman. A Retrospective View of the HEARSAY-II Architecture. In *Fifth International Joint Conference on Artificial Intelligence (IJCAI'77)*, pages 790–800, Cambridge, Massachusetts, August 1977.
11. 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.
12. J. Myplopoulos and M. Papazoglu. Cooperative Information Systems, Guest Editors' Introduction. *IEEE Intelligent Systems*, 12(5):28–31, September/October 1997.
13. H. P. Nii. Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Magazine*, 7(2):38–53, 1986.
14. A. Preece, S. Chalmers, C. McKenzie, J. Pan, and P. Gray. Handling Soft Constraints in the Semantic Web Architecture. In *Reasoning on the Web Workshop (RoW2006), in the World Wide Web Conference (WWW2006)*, Edinburgh, UK, 2006.
15. 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.
16. 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.
17. M. Wooldridge. *An Introduction To MultiAgent Systems*. Wiley, 2002.