Implementing a Semantic Web Blackboard System using Jena

Craig McKenzie, Alun Preece & Peter Gray
{cmckenzie,apreece,pgray}@csd.abdn.ac.uk
Department of Computer Science, University of Aberdeen, Aberdeen AB24 3UE, UK

Abstract

In this paper, we discuss the need for a hybrid reasoning approach to handing Semantic Web data and explain why we believe that the Blackboard Architecture is particularly suitable. We describe how we have utilised it for combining ontological inference, rules and constraint based reasoning within a Semantic Web context.

After describing the metaphor on which the Blackboard Architecture is based we introduce the key components of the architecture: 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 Semantic Web Blackboard System in Java, using Jena. We also discuss our reasons why we elected to use the Jena toolkit and explain its usage within several of the key components of our system.

1 Introduction & Motivation

Our research interest lies in exploring the suitability of a Blackboard System to utilise incomplete, Semantic Web 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[1] [10], namely the Computing Science (CS) community in the UK. Our 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 an interest of *Agents*". Having successfully used Jena in past developments, our familiarity with the toolkit meant that we knew it was capable of handling our initial requirements and, as we discuss later, does not appear to restrict us in our future work plan either.

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 [11] when a DL based reasoner was compared to a First-Order

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

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 [1] 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 [9, 2].

The structure of this paper is as follows. In section 2 we introduce our test-bed application, the AWB+B, and explain the process of building workgroups. Then, in section 3, we describe the blackboard analogy before comparing the traditional approach to our Semantic Web based approach. In section 4 we describe the role of the KSs within the architecture and discuss their individual attributes using the AWB+B to help illustrate the concepts. Next, section 5 describes the controlling mechanism of the blackboard. Finally, in section 6 we conclude with a discussion of our findings and comment on the direction of our future work.

2 Building Workgroups

The AKTive Workgroup Builder and Blackboard (AWB+B) is a new incarnation of our earlier version of the system (AWB [8]) that also used Jena but 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 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 work it out 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

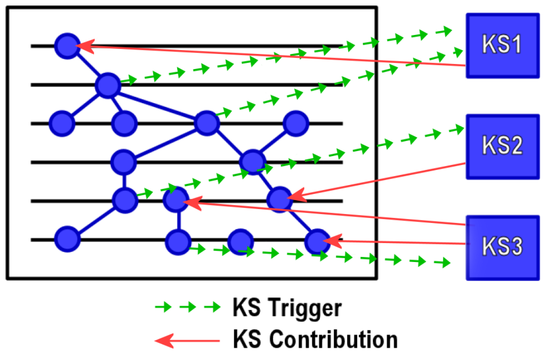

Figure 1: Each Knowledge Source (KS) can view the Abstraction Levels within a Blackboard Panel. A KS can be *triggered* by any of the items on blackboard, allowing it to *contribute* something at any of the abstraction levels.

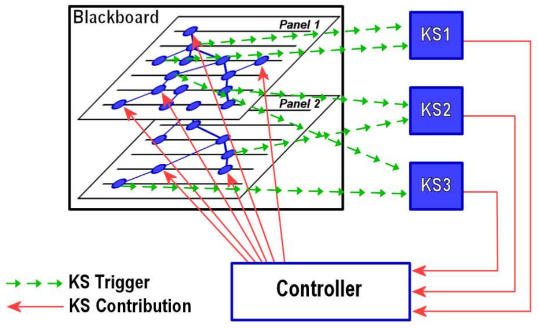

Figure 2: The core architectural components of a Blackboard System. Each KS can view the contents of the Blackboard Panels, but it is the Controller that decides which KS(s) are allowed to contribute to the Blackboard.

fuller description see [9]). No-one is allowed to communicate directly, everything must be done through the blackboard which becomes a shared "thinking space" for all the participants.

In computing terms, the architecture of a Blackboard System has the "blackboard" as a shared knowledge base, and the "people" as various KSs – we discuss KSs in more detail in section 4.

3.1 Traditional Approach

The pioneering blackboard systems (e.g. Hearsay-II [4], HASP/SIAP [5], CRYSALIS [3] and OPM [7]) 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 of the layer below it. For example, using the domain of speech understanding, 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 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 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 the AWB+B it is a Jena `OntModel` that is used to store the data. Our decision to use Jena was influenced by our familiarity with the toolkit but, primarily, rather than just being a *triple cache* for RDF data, Jena uses a graph representation, with an intuitive and convenient API for accessing content. 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).

Until now, the blackboard has been passive, with any reasoning function placed firmly on the shoulders of the KSs. While not wishing to stray too far from the original concepts of the blackboard architecture, we have introduced an element of intelligence to the blackboard itself. 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. Unfortunately, this did become an issue when we attached an OWL Lite reasoner. Therefore, we elected to only infer transitive sub-class/property relations on classes and instances. Because this is such a common operation, having it done by the blackboard eliminates the need for frequent call outs to KS that would perform the same function. In implementation terms, we simply attached a `GenericRuleReasoner` and the following set of 4 forward chaining rules to the `OntModel`:

```
(?a rdfs:subPropertyOf ?b),(?b rdfs:subPropertyOf ?c) -> (?a rdfs:subPropertyOf ?c)
(?a ?p ?b), (?p rdfs:subPropertyOf ?q) -> (?a ?q ?b)
(?a rdfs:subClassOf ?b), (?b rdfs:subClassOf ?c) -> (?a rdfs:subClassOf ?c)
(?x rdfs:subClassOf ?y), (?a rdf:type ?x) -> (?a rdf:type ?y)
```

Jena is quite flexible in that there are a choice of their own rule based reasoners that can be attached but also because Jena’s architecture provides a mechanism for attaching external reasoners to Jena models (by implementing the DIG description logic reasoner interface). This means that, down the line, our research will not be restricted as we can incorporate reasoners such as FaCT[2], Racer[3] or Pellet[4].

[2]`http://www.cs.man.ac.uk/~horrocks/FaCT/`
[3]`http://www.sts.tu-harburg.de/~r.f.moeller/racer/`
[4]`http://www.mindswap.org/2003/pellet/index.shtml`

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 black box. Due to the tightly coupled nature of the KSs and the Blackboard, all KSs must be "registered" so that they can view the blackboard contents and inform the Controller of any potential contributions.

KSs 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 governed by a controller which 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.

4.1 Human (User Interface) 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 a web-based user interface. In AWB+B terms this is the user specifying the problem parameters, e.g. the number of workgroups to be built, the size of each workgroup, any associated 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, thus kick-starting the 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 Human 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 only contains instance data corresponding to an ontology but not the actual schema itself. This could either be from a simple RDF file or be 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 defintion information about the `Lecturer` or `University` classes.

If this KS is a repository of RDF triples (for example, 3Store [6]) 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[5] queries. We transform any blackboard goal into a query, the result of which can be transformed into triples and asserted onto the blackboard.

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, 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) 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:
   - 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 or 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, would be 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). 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).

[5]SPARQL (SPARQL Protocol And RDF Query Language) is a query language for getting information from RDF graphs. The W3C working draft, *SPARQL Query Language for RDF*, is available at: `http://www.w3.org/TR/rdf-sparql-query/`

4.4 Rule Engine 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[6]. 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[7]. 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 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 and/or properties. 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) that could, subsequently, cause other KSs to add irrelevant information relating to these (either solution instances or sub-class/property sub-goals).

The current AWB+B implementation has only one starting rule per Rule KS. The rule is rewritten into a SPARQL query, which is placed against the blackboard contents to determine if any new triples can be asserted. The results of the query are asserted onto the blackboard as the new triples. This is, essentially, a brute force, forward chaining approach to deriving new entailments (and goes against the grain of the backward chaining approach of the blackboard).

In our future work we plan to further investigate the effects of Rule Chaining and plan to take advantage of Jena's rule engine by converting the SWRL rules into the Jena format and applying these directly to the blackboard. This tighter integration, coupled with a backward chaining approach would greatly improve efficiency. Since the KSs access the blackboard via an API, we have the alternative option of using an external rule engine too (e.g. Prolog, Jess[8], SweetRules[9]).

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 [8] – our Constraint Interchange Format (CIF), which is an RDF based extension of SWRL that

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

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

[8]`http://herzberg.ca.sandia.gov/jess/`

[9]`http://sweetrules.projects.semwebcentral.org/`

allows us to express fully quantified constraints. Since the goal of the AWB+B is to form workgroups that adhere to the specified constraints, this KS is integral to the system and causes the termination of the goal, i.e. once a solution to the problem has been found, then there is little point in continuing!

There are essentially two ways for this KS to operate. The first method is the easiest to implement. It involves the Controller waiting until all the other KSs have finished contributing and the contents of the blackboard are as complete as they can be. Only then is a solution checked for. The second method involves the KS continually checking if a solution can be found. Rather than attempting full blown CSP solving continuously, the solver checks each of the constraints individually and only if they can all be satisfied, does it attempt the more difficult task of solving them combinatorially. It is the first approach which we have adopted in the AWB+B.

If a solution is found the workgroups are posted onto the blackboard. For convenience, the user is also presented with a web page listing the proposed members of the workgroup(s). Otherwise, they are informed that no solution can be found. In the latter case, the user must decide which constraint(s) must be relaxed before the problem can be attempted again. The system aids this decision by also including unsatisfiable constraint combinations.

5 The Controller

As the name suggests, the role of the Controller is to oversee the running of the system as a whole. So far we have talked about the contents of the blackboard as merely containing the 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 "instances of `<owl:Thing>`" to be placed onto the blackboard. Since this is *the* OWL super-class and would result in *all* class (and sub-class) instances known by a KS being added onto the blackboard. KS access to the blackboard is via a restrictive API that allows only those `OntModel` method calls that view the underlying graph but not modify it. This is in place to prevent a KS from modifying the blackboard 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 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 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 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.

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, as well as introduce threading to improve concurrency (taking advantage of Jena's `ModelChangedListener` interface).

6 Conclusions

In this paper we have discussed the implementation of a blackboard system in Java, using Jena. From the onset, we wished to make the system as generic as possible, allowing groups and constraint reasoning to be orthogonal to the choice of ontology by not giving the underlying blackboard any ontological structure.

An important issue we encountered was a lack of efficiency of the two stage approach of `canContribute` and `makeContribution` for KS interaction; the work involved in determining if a KS can contribute is comparable to the actual work involved in making the contribution. 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. We would argue that performance could be improved by committing more time to the development of the AWB+B to further optimise it.

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 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. In our future work we plan to investigate complexity and scalability trade-offs (e.g. using multiple ontologies that require being mapped, increasing the size of the dataset, etc.) as well as the modification of the Controller strategy to enhance performance.

Even with these improvements, we still foresee that the time taken to solve the CSP can exceed the acceptable threshold for rendering a web page in real-time. Quite simply, reasoning and CSP solving are time consuming, so a certain level of patience is expected on the part of the user. Even with application optimisation, there are still complexity issues. The Blackboard framework gives us a number of options on how to explore these in our future work. What if the data on the blackboard makes the CSP undecidable? What happens if a KS starts to perform reasoning that could take hours or days to complete? Thankfully, the Blackboard Architecture guards against the inefficiency of KSs – 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 to the Controller, so if a KS takes an inordinate amount of time to respond it is just ignored. The architecture supports the addition or removal of KSs from the system, with the only adverse effect being on quality of the results.

Acknowledgements
This work is supported under the Advanced Knowledge Technologies (AKT) IRC, supported by the EPSRC (grant number GR/N15764/01). The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. For further information see: `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. CMP-SCI Technical Report 92-71, Computer Science Department, Southern Illinois University, 1992.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.