AKTive Workgroup Builder: Semantic Web Instance Data Reasoning

Craig McKenzie, Alun Preece, and Peter Gray

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

Abstract. Our interest lies in exploring the interplay between ontological and rule based reasoning with instance data when applied to Constraint Satisfaction Problem (CSP) solving. The AKTive Workgroup Builder (AWB) is a Semantic Web application developed to help us achieve this aim. We share our experiences of developing the AWB – how the current technologies can influence the usability and design of such an application – and describe our approach to reasoning using both ontological and rule based methods. We show how these rules can be represented using the Semantic Web Rule Language (SWRL). Constraints are then expressed against the semantic data using our Constraint Interchange Format (CIF) combined with SWRL to form a fully quantified constraint representation CIF/SWRL. Finally, the problem specific constraints and the reasoned domain knowledge are then bundled together into a CSP which the AWB attempts to solve, returning the solution (if there is one) to the user.

1 Introduction & Motivation

As more and more semantically marked up information becomes available, the majority of Semantic Web (SW) applications tend to offer some form of query or navigation service. In line with the vision that 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 [3], we required a tangible, SW problem that incorporates all these elements. Since our interest lies in investigating reasoning with SW instance data and applying it to a constraint based problem – specifically, the interplay of ontological inference and rule based reasoning along with (finite domain) constraint solving – it seem fitting to apply this to a construction/configuration problem. An interesting starting domain was with the context of the CS AKTive Space[1] [6], namely the Computing Science (CS) community in the UK. Our demo application, the AKTive Workgroup Builder (AWB), attempts to construct a workshop, containing one or more working groups of people from a pool of known individuals, that adheres to a set of

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

user defined constraints. The pool of individuals is created dynamically based upon semantically marked-up information from known, distributed data sources, where the quality (and completeness) is not guaranteed (discussed later).

In this paper, we start by describing the architecture of the AWB and then discuss the factors that influenced the application's design. Initially, we did not foresee the need for using rule based reasoning – we assumed that ontological based reasoning alone would be adequate – however, as our work progressed it became apparent that deriving new information through rules is an extremely important and powerful asset. For this we describe our use of the current proposal of the Semantic Web Rule Language[2] (SWRL) to express these logic based derivation rules. We then demonstrate how remapping these rules into another form allows their application to our dataset.

To facilitate this, we argue for the importance of not only making available semantic constraint information [2] but also the ability to express this in a fully quantified and natural manner. Building on our previous work with our Constraint Interchange Format (CIF) [4] we demonstrate how this can be achieved using CIF/SWRL[3], an extension of SWRL with our RDF compatible encoding of CIF [5]. Derivation rules and constraints belonging to the data can be considered not only as important descriptors of that data but also as indicators of how to use or validate the data.

We conclude with a discussion of our findings and comment on the direction of our future work.

2 Building Workgroups

The AWB attempts to solve the problem of assembling one or more workgroups from a pool of known people. A workgroup is simply a solution set of individuals whose membership satisfies all the constraints applied to it to restrict its formation. A user constructs a workgroup by following these steps (see figure 1):

1. Information about the pool of people to be considered is gathered from relevant datasource(s);
2. Quantified constraints are specified by the user about the composition of the workgroup(s), e.g. minimum and maximum size, the focus, etc.;
3. Reasoning is performed against the data to determine eligibility, e.g. does a person have the relevant research interests and can those interests be determined if not explicitly stated?
4. Finally, the constraint satisfaction problem is built and then passed on to a solver that attempts to compose workgroup(s) that satisfy the stipulated constraints.

Implemented as a Java Server Pages application, the AWB uses Jena[4] to manage the RDF processing and some of the reasoning, with SICStus Prolog being used for the Finite Domain (FD) Constraint Satisfaction Problem (CSP) solving.

[2] `http://www.w3.org/Submission/SWRL/`
[3] `http://www.csd.abdn.ac.uk/research/akt/cif/`
[4] `http://jena.sourceforge.net`

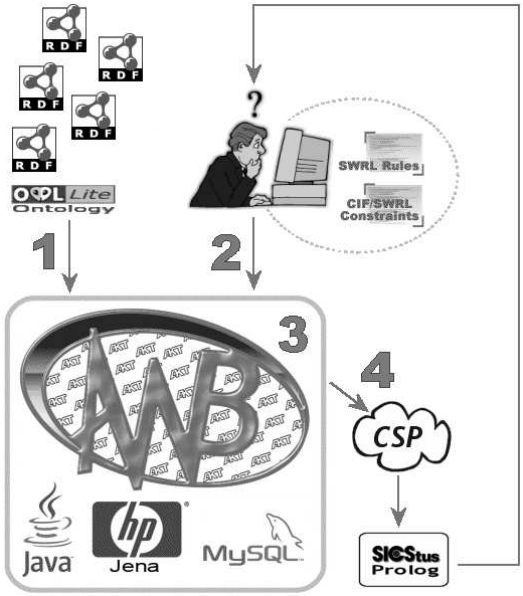

Fig. 1. Steps involved in workgoup building, using the AWB.

The RDF data processed by the AWB contains, among other things, 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. This information will then be reasoned against to infer additional facts about the individuals that may not have been explicitly stated - for example projects that they have worked on or papers that they have published can imply additional research interests.

When building a workgroup, the user of the application specifies explicit constraints pertaining to the nature of the workgroup (for example, the workgroup has the focus of "Machine Learning" and cannot contain more than 10 persons). In addition to these, there are further implicit constraints that are also applied (for example, a workgroup must contain at least two people and a person can only be a member of one workgroup at a time).

The data and its associated constraints are passed on to a constraint solver (Prolog) so that a suitable workgroup (or workgroups) can be composed satisfying all these specified constraints. Once a solution has (or has not) been reached, the user will be presented with the proposed members of the workgroup(s) or informed that no solution can be found.

The time taken to gather information and perform reasoning falls outwith the time deemed as acceptable to serve a Web page in real-time. To overcome this hurdle, the application was designed to create a pre-reasoned triple-cache

stored in a MySQL[5] backend database, populated prior to workgroup building. We recognise that this is not ideal, especially because a single point for serving what was previously distributed data can cause problems. Data ageing is an important consideration, so we store provenance information describing the originating source and the date when the 'snapshot' was taken.

This forced the user interface to be split into two functional parts - an *admin* aspect where the database information is constructed and maintained, and a *build* aspect that allows the user to specify which database model to utilise as well as the workgroup composition constraints before composing the CSP and then starting the solver process.

Unfortunately, both the cache creation and the workgroup building are a heavy burden on the system resources (monopolising the CPU and easily taking upwards of 20 minutes depending upon how heavily constrained the workgroups are), so the implementation of a 'call back' mechanism was necessary. This allows the user to start a process running and has the option of continuously monitoring progress, or 'forgetting about it' and checking their *user messages* for progress/completion information later on.

3 Reasoning with Instances

Utilising SW data has the fundamental problem of locating a datasource and ensuring that the provided information is usable in practical reasoning. Initially we had planned to access the CS AKTive Space repository directly, however the information contained within it is against the AKT Portal ontology, which is OWL Full, and since there is a lack of OWL Full reasoning support (as well as it having no guarantee of decidability), we were forced to transform the AKT Portal ontology to remove the OWL Full constructs.

The major part of the AKT Portal ontology was OWL Lite, and we began by isolating the portion of this supported by the reasoner in HP's Jena toolkit (which we intended to use for RDF manipulation and reasoning). The changes to the ontology definitions were relatively minor, mostly weakening to adhere to OWL Lite; for example, `<owl:oneOf>` and `<owl:unionOf>` are not supported. We also made some pragmatic changes to the ontology; a flattening of the hierarchy tree and a simplification of the publication related branch of the taxonomy - an issue recognised by the CS AKTive Space project and commented upon in their paper [1].

The data contained within the CS AKTive Space repository was scraped from HTML pages with other information being hand crafted RDF. This has resulted in the content of the repository being incomplete and containing errors, contradictions and duplications. The sheer scale of the amassed data causes provenance and trust issues which we decided to avoid tackling at this stage – meaning we required a 'tidy' dataset based on the existing information. In our early experiments the sheer volume of data (10M+ triples) also impeded our progress – due

[5] See: `http://www.mysql.com`

mainly to the reasoning time. Therefore, as a starting point we used a subset of manually, cleaned-up data that was manageable, yet still realistic, and also ensured tractable reasoning. This set only covers the individuals involved in the Advanced Knowledge Technologies (AKT) project.

This 'sandbox' enabled us to make several assumptions about the environment we are working in and focus primarily on the interplay between rules, constraints and instance based reasoning rather than fire-fighting data and/or ontological reconciliation issues. Ontological reasoning involves generating new facts based upon the properties and class hierarchy within the ontology. Therefore, we assume that the underlying ontology is consistent and well formed and that the associated instance data is valid (i.e. does not contain contradictions) and any equivalences are explicitly stated.

In building the cache, we effectively pre-compute all RDF(S) types based on the facts and asserted rules we have (because OWL Lite is decidable) and the derivation rules assert 'safe' factual information. We believe that this is possible because we only have a finite domain of instance data.

4 Derivation Rules

Although a reasoner can derive additional ontological entailments, based upon property and class hierarchies, sometimes it is necessary to infer pertinent information that cannot be determined otherwise, hence the need for derivation rules.

SWRL is used to state these rules, since it is based upon OWL Lite and DL and therefore has close ties to the underlying ontology. It allows Horn clauses to be asserted about the semantic data to create implications which we use to encode our derivation rules (making them available along with the data). We utilise these to create new facts based on the instance data. For example, consider the following rule, used to determine the `hasBaseLocn` property: *"If a person has an affiliation with a university and that university has a postal address of a city, then this implies that the person has a base location of the same city where the university is located."* In informal SWRL syntax, where $?x$ denotes a variable, this can be written as:

$$\texttt{Person}(?p) \wedge \texttt{University}(?u) \wedge \texttt{hasAffiliation}(?p,?u) \wedge \texttt{hasAddress}(?u,?c) \wedge \texttt{City}(?c) \Rightarrow \texttt{hasBaseLocn}(?p,?c)$$

In the AWB we have the option of adopting two different approaches to generating the derived rules. The first method is the *greedy* approach, where as many entailments are derived as possible (forward-chaining). One possible implementation of this 'brute force' method involves mapping the rules into RDQL queries, the results of which are added back into the knowledge base. For example, using the same `hasBaseLocn` derivation rule we can create the following query, where `rdf` is the RDF namespace, and `ont` is the appropriate ontology namespace:

```
SELECT ?person, ?city
```

```
WHERE (?person, <rdf:type>, <ont:Person>),
      (?uni, <rdf:type>, <ont:University>),
      (?person, <ont:hasAffiliation>, ?uni),
      (?uni, <ont:hasAddress>, ?addr),
      (?addr, <rdf:type>, <ont:City>)
```

The result set of this query contains value pairs of *Person* URI and *City* URI which could easily be asserted into the datastore as a new triple:

```
?person <ont:hasBaseLocn> ?city .
```

Since we could be applying multiple rules, this is an iterative process, with the entire sequence repeated until no new entailments are generated. This (rather inefficient) repetition is necessary as the interdependency of multiple rules cannot easily be determined. This has the disadvantage of having to perform unnecessary processing, but subsequent queries are faster as the facts now exist in the knowledge base and do not have to be recalculated in future.

The second method could be thought of as *lazy* derivation where only those entailments that are required to be generated are used (backward-chaining) and, hence, is quite efficient. The AWB implements this by translating the rules into Prolog and passing them, along with the data, onto the CSP solver. This approach means that no unnecessary processing is performed and dependencies are handled easily within Prolog. This method would be far more preferable in future when the overall reasoning time drops and optimisation is important. The disadvantage of this is that the rules are fired for each new instance of the CSP. To overcome this, it is necessary to extract the newly derived facts (in this case, any newly discovered base locations) back from the CSP solver in order to update the database cache after the solution space has been searched.

5 Semantic Web Constraints

A gap exists in the Logic layer of the SW because there is currently no standard method for expressing fully quantified constraints against semantic data in a natural manner [5]. In the context of the AWB this means being able to pass user specified constraint information relating to the construction of a *particular* workgroup or *all* workgroups. Such a representation is important because it allows constraint information to be made available along with the data itself – potentially allowing a partially solved problem to be passed onto a solver or providing provenance information about how a solution is constrained.

Unfortunately, when we started work the original SWRL proposal was limited in that fully quantified constraints could not be expressed[6], therefore we elected to extend SWRL with our Constraint Interchange Format[4] (CIF) in order to achieve this. The CIF representation is based on range restricted First

[6] The SWRL-FOL W3C Member Submission introduces quantifiers but is still under discussion and, moreover, the lack of RDF syntax makes it unsuitable for use by us at this stage. See: `http://www.w3.org/Submission/2005/SUBM-SWRL-FOL-20050411/`

Order Logic (FOL), and has evolved to use RDF(S). CIF constraints are transformable for use with a variety of constraint logic programming solvers, including CHIP, ECLiPSe, and the SICStus Prolog FD library [5]. The AWB uses constraints defined by the user to control the construction of the workgroup. The following example shows how a constraint can be expressed using CIF/SWRL [5] and draws upon the previously specified `hasBaseLocn` property: *"Any workgroup containing at least five members must contain at least two individuals from differing sites (base locations)."* This can be written as:

```
(∀ g ∈ Workgroup) hasSize(g,s) ∧ greaterThanOrEqual(s,5)  ⇒
  (∃ p1,p2 ∈ Person) hasMember(g,p1) ∧ hasMember(g,p2) ∧
    hasBaseLocn(p1,b1) ∧ hasBaseLocn(p2,b2) ∧
    notEqual(p1,p2) ∧ notEqual(b1,b2)
```

Further examples, in RDF syntax, are given in [5].

The AWB uses the SICStus Prolog FD Constraint Library[7] in the CSP solving and therefore makes a closed world assumption (with negation-as-failure). In the CSP construction we are effectively adopting a 'best information at hand' approach, for example, if we had the constraint "*a workgroup must not contain anyone who is a lecturer*" then the set of valid (non-lecturer) candidates would be compiled based on the difference with the set of known lecturers. While this might seem contradictory to the open world assumption of the SW in general, we have found that this still delivers valid results in practice.

6 Discussion & Conclusion

In its present form, the AWB provides a demonstration of a practical SW problem-solving system that uses a mix of reasoning methods on instance data: ontological entailment, derivation rules, and finite domain constraint solving. We believe that in this respect the AWB represents a novel contribution. This section highlights lessons learned from the AWB work to date.

Pragmatic issues of data gathering, computational cost of reasoning, and data quality lead us to pre-cache the instance data on which the AWB would operate. While this caching model is far from ideal, we are able to take advantage of the fact that reasoning is time consuming by using derivation rules to pre-generate entailments (via forward chaining) without any real impact on the user. Since this is performed prior to the AWB user accessing the cache and specifying the constraints relating to the workgroup they wish to build, many of these generated entailments can merely be thought of as 'attempting to cover all bases' since they may never be used at all. Fortunately, the AWB also has the luxury of using derivation rules at runtime (rewriting SWRL rules into Prolog predicates and including these into the CSP). This uses backward chaining and means that the rules are only fired if they are needed. However, inclusion of these predicates

[7] `http://www.sics.se/isl/sicstus/`

must be done sparingly as the time taken to solve the CSP can then exceed the acceptable threshold for rendering a single web page. Should this happen, we must then resort to a "call back" message informing the user once the solution has been found. While this implementation allows us to side-step a potential usability problem, the underlying issue still stands. The AWB has been designed to allow the exploration of the trade-offs, in practical terms, of the effectiveness of the different reasoning approaches (e.g. greedy vs. lazy, forward vs. backward chaining) and this work is ongoing.

Future work will be to extend the functionality to cater for the scenario when no solution can be found, requiring that certain constraints must be relaxed (and informing the user appropriately). There is also the interesting possibility of opening a 'negotiation dialogue' to achieve a solution when several workgroups are being built at the same time and resources are restricted.

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

References

1. H. Glaser, H. Alani, L. Carr, S. Chapman, F. Ciravegna, A. Dingli, N. Gibbins, S. Harris, m. schraefel, and N. Shadbolt. CS AKTiveSpace: Building a Semantic Web Application. In C. Bussler, J. Davies, D. Fensel, and R. Studer, editors, *The Semantic Web: Research and Applications (First European Web Symposium, ESWS 2004)*, pages 417–432. Springer-Verlag, 2004.
2. P. Gray, K. Hui, and A. Preece. Mobile Constriants for Semantic Web Applications. In M. Musen, B. Neumann, and R. Studer, editors, *Intelligent Information Processing*, pages 117–128. Klewer, 2002.
3. J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, pages 30–37, March/April 2001.
4. K. Hui, P. Gray, G. Kemp, and A. Preece. Constraints as Mobile Specifications in e-Commerce Applications. In *Proceedings of the 9th IFIP 1.6 Working Conference on Database Semantics (DS-9): Semantic Issues in e-Commerce Systems*, pages 357–379, 2001.
5. C. McKenzie, A. Preece, and P. Gray. Extending SWRL to Express Fully-Quantified Constraints. In G. Antoniou and H. Boley, editors, *Rules and Languages for the Semantic Web (RuleML 2004)*, pages 139–154, Hiroshima, Japan, 2004. Springer.
6. 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.