Using Semantic Web Languages in Argumentation Models

Florian Echtler

TU München <echtler@in.tum.de>

Technical Report FKI-253-06
AI/Cognition Group
Technical University of Munich

Abstract. Recent research has created a multitude of argumentation models with varying degrees of formality. In this paper, we look at the feasibility of using semantic web languages like OWL and SWRL as an unbiased template for developing such models. We then present the Argumentation Ontology (ArgOn) as an example.

1 Introduction

As seen in the comprehensive survey by Rahwan et al. [Rah03], a large amount of papers regarding argumentation-based negotiation (ABN) have been published in recent years. Most of these publications formalize the subject to some extent, however, this presents several challenges.

These formal models, for example, are usually not very similar and therefore difficult to build on.[1]

When an actual implementation is desired, the specification has to be hard-coded into the negotiating software agents and is consequently hard to change.

Moreover, the agents usually communicate in a different formal language than the one which was used for the argumentation scheme.

All these shortcomings can be addressed in terms of an ontology of argumentation. Such an ontology has to be sufficiently neutral to subsume the majority of existing argumentation models. When a model has now been re-formulated, a suitable parser can afterwards provide easily adaptable source code. Additionally, both the definition of the dialogue and its content are now described using with only a single formalism.

[1] Interestingly enough, no single one of these models has so far emerged as *the* argumentation standard. This suggests the conclusion that argumentation models have to be tailored towards a specific application.

2 The Argumentation Ontology (ArgOn)

The slowly emerging standard for ontologies is OWL [McG04], the Web Ontology Language. Unfortunately, OWL lacks the expressive power to represent the modal logics which are usually employed to represent argumentation models, because it is impossible to create new (modal) operators without violating the language specification.

However, two extensions to OWL have been proposed, SWRL [Hor04] and SWRL-FOL [Pat05], which allow the expression of Horn-like rules and first-order logic, respectively. Along with *reification* (meta-statements, described more in-depth below), it is possible to achieve expressive power similar to existing modal logic systems.

OWL itself is an extension of the RDF (Resource Description Framework), which is in turn an XML-based language to formalize statements as triples of subject, predicate and object. From a formal point of view, RDF describes a directed graph, where subjects and objects represent the nodes, while predicates stand for edges from subject to object (see also [Kly04]).

OWL comes in three flavours, which are OWL Lite, OWL DL and OWL Full and increase in expressiveness as well as computational complexity regarding conclusions. It would be desirable for the ontology to remain inside OWL DL, where all conclusions are guaranteed to be decidable. Unfortunately, as reification is necessary to model argumentation, the expressive power of OWL DL is insufficient. The presented ontology is therefore in OWL Full.

As a development tool, the Protege editor with additional plugins for OWL and SWRL was chosen. An overview of the ontology structure is presented in Figure 1. There are three main concepts, which will be detailed below.[2]

2.1 Core Concepts Relating to Agents

There are two main classes with regard to agents, `Resource` and `Actor`. It is assumed that each individual which takes part in the argument has its own knowledge base.

The `Actor` Class. Actors represent participants in the negotiation process. Remember that each agent holds its own instance of the knowledge base, therefore it contains an instance of Actor for every known participant including itself. This latter instance is marked by an auxiliary predicate `self` of datatype `xsd:boolean`.

Actors are described through four additional predicates:

[2] For better readability, class names will be set in `monospace font`.

`canDo` describes the actor's abilities (instances of `Ability`)
`owns` is a list of the actor's possessions (instances of `Object`)
`knows` contains the knowledge of the actor (instances of `Information`)
`does` is a history of things the actor has done (instances of `Action`)

Note that in most cases, these properties contain information about some other actor which may therefore be incomplete, as no agent is omniscient.

A multiagent system based on this ontology should take care that each participant can be referred to by an unique identifier. This is necessary to make unambiguous statements about other agents.

The `Resource` Class. Resources represent a generic view of those entities that the argument is about. Each may require and/or consume other resources, e.g. performing an action usually consumes time (which is also represented as a resource) and may require certain abilities.

An utility value is associated with every resource an agent knows about. It is assumed that every agent aims to maximize its own total utility, thereby defining its goals in the negotiation.

The `Resource` class has four subclasses:

`Ability` Abilities are prerequisites for actions. There are two default abilities which every actor is capable of, `Communicate` and `Nothing`. Abilities can be transferred between actors.
`Action` An instance of a subclass of Action represents one single execution of this action at a specific time.
`Object` represents a passive physical (e.g. an apple) or virtual entity (e.g. bandwidth).
`Time` models a timespan, for example the time required to execute an action.

Of these abstractions, the most important one is `Action`, which shall be discussed in detail. The subclasses are designed to be as generic as possible, in order to put the least possible influence on further development. An action is described, in addition to the predicates inherited from `Resource`, by `happensAt` (as this is a single event, it occurs at a specific time) and `changes`. This latter predicate describes the properties that are changed when this action is executed.

First, the actions `Create` and `Idle` should be mentioned, which model the construction of new objects from existing ones and the null-action, respectively. Creation of a new object, for example, `changes` the property `owns`, as the creator afterwards has this new entity in its possession.

Of far greater importance, however, is the `Transfer` action. It has three subactions, which are `Give` (transfer of `Objects`), `Teach` (transfer of `Abilities`) and finally `Tell` (transfer of `Information`).

`Tell` is undoubtedly one of the most central classes of this argumentation ontology, as the argumentation itself is but a transfer of arguments, which are in turn pieces of information.

It might therefore seem questionable to locate this class several levels deep inside the class hierarchy. However, as any action can itself be a resource which is negotiated over, and the transfer of information is in turn a more specific action than the transfer of abstract entities, this nesting can be justified.

2.2 Core Concepts Relating to Argumentation

The `Information` Class. Information instances are used similar to `rdf:Statement` objects to represent propositions (reification, meta-statements)[3]. Like its predecessor, this class has the three primary predicates `subject`, `predicate` and `object`.

From an abstract point of view, such an object now describes one edge in the RDF graph which constitutes an agent's knowledge base. As this is therefore a meta-statement, the ontology expands to a superset of OWL DL and the guaranteed decidability is lost. However, without reification, expressing modalities like "Statement S is believed by actor A" becomes impossible. As modality of beliefs is absolutely necessary to describe argumentation - there is no need to argue if everyone has the same beliefs - the loss of guaranteed decidability is unfortunate but unavoidable.

Again, several subclasses have been defined to refine the concept of `Information`, for example `Request`, which encapsulates the concept "`Actor does Action`". This can be refined further, e.g. into a `Question`, where the action is defined to be `Tell` (see below for an example).
A different problem arises from the fact that OWL is based on the open world assumption - every statement in the knowledge base is true, and all other statements are unknown. In argumentation, however, one must be able to express that a certain statement is *not* true. For this purpose, a new auxiliary predicate `truthValue` is used, which describes the beliefs of the agent from which the statement originated. It is a data-valued property of type `xsd:boolean` and is introduced in a subclass `Argument`.

By using this predicate, an agent can now state that he believes a certain information to be false, in order to influence the world view of some other agent. It is, of course, up to the receiving agent to decide whether to accept the new statement and, consequently, delete the corresponding edge from its RDF graph.

[3] `rdf:Statement` itself can't be used as SWRL only allows using OWL entities in rules, not RDF entities.

Should it decide to disregard the information, it can still insert the received statement into its own representation of the other agent through the `knows` property. This knowledge can be used later, for example, to notice inconsistencies in the argumentation flow.

2.3 Using SWRL Rules

The parts of the ontology described so far deal with describing agents, statements, and resources. However, they do not describe the behaviour of the agents themselves. As OWL itself is static in nature, the necessity for a rule language becomes obvious.

Though SWRL is so far no W3C recommendation, it is a well-fitting extension to OWL. It allows the creation of Horn-like rules, consisting of the conjunction of a set of atoms as rule body and usually a single atom as rule head. An atom usually takes the form `Class(?variable)` or `Predicate(?var1, ?var2)` and is true when the individual in question is a member of the specified class or a predicate connecting the two individuals exists.

Consider the following example. It describes an agent which will do everything it is asked for without disputing the request.

```
   Request(?request) ∧ object(?request, ?action) ∧
subject(?request, ?actor) ∧ self(?actor, true) ⇒
does(?actor, ?action)
```

2.4 Example Dialogue

In this section, some snippets of a possible conversation between two agents Alice and Bob are presented. These snippets are given in RDF triple notation, with a namespace prefix of `ex:` for example.

The following triples are supposed to be present and unchangeable in the knowledge bases of all agents. This should be guaranteed by the framework, for reasons already discussed in Section 2.1.

```
ex:Alice  rdf:type    ex:Actor .
ex:Bob    rdf:type    ex:Actor .
```

Let us assume that Alice believes Bob to have something she wants. The following triples are in her knowledge base:

```
ex:Alice  ex:self     "true"^^xsd:boolean .

ex:Item1  rdf:type    ex:Object.
ex:Item1  ex:utility  "2.0"^^xsd:float .
```

```
ex:Item2  rdf:type    ex:Object.
ex:Item2  ex:utility  "3.0"^^xsd:float .

ex:Alice  ex:owns     ex:Item1 .
ex:Bob    ex:owns     ex:Item2 .
```

Now, consider Alice asking Bob to give her Item2, which she values more highly. This snippet of RDF is created in Alice's knowledge base. Ideally, the framework should take care of transmitting it to Bob after the `Ask` object has been created.

```
ex:act1   rdf:type      ex:Give .
ex:act1   ex:recipient  ex:Alice .
ex:act1   ex:content    ex:Item2 .

ex:req1   rdf:type      ex:Request .
ex:req1   ex:subject    ex:Bob .
ex:req1   ex:predicate  ex:does .
ex:req1   ex:object     ex:act1.

ex:msg1   rdf:type      ex:Ask .
ex:msg1   ex:content    ex:req1 .
ex:msg1   ex:recipient  ex:Bob .
```

However, what if Alice's information is outdated and Bob doesn't have the item in question any more? Bob will refuse the request and might also want to give a reason for the refusal, as Alice would otherwise still wrongly believe that Bob is in possession of Item2. The following statements are therefore sent back from Bob:

```
ex:arg1   rdf:type      ex:Argument .
ex:arg1   ex:subject    ex:Bob .
ex:arg1   ex:predicate  ex:owns .
ex:arg1   ex:object     ex:Item2 .
ex:arg1   ex:truthValue "false"^^xsd:boolean .

ex:rej1   rdf:type      ex:Rejection .
ex:rej1   ex:subject    ex:req1 .
ex:rej1   ex:predicate  ex:reason .
ex:rej1   ex:object     ex:arg1 .

ex:msg2   rdf:type      ex:Reject .
ex:msg2   ex:content    ex:rej1 .
ex:msg2   ex:recipient  ex:Alice .
```

3 Discussion

3.1 Comparison with Existing Approaches

Verheij [Ver05] also presents a draft of an argumentation ontology, written in OWL. However, this ontology has several shortcomings. Its class hierarchy seems quite arbitrary, and beyond the subclass relation, no further connections between the different argument types are described, neither by using reification nor through rules. Moreover, it doesn't touch the agent aspect which should be present in any practice-oriented approach to argumentation, as an orientation towards practice is already implied when using machine-readable languages like OWL.

3.2 Future Work

The ontology presented in this paper is intentionally rather basic, in order to remain as unbiased as possible. Therefore, it provides several opportunities for future research.

As one of the goals of this work is to provide an easy comparison between different argumentation models, it would be necessary to represent existing and emerging models in terms of the ontology.

When working with complex argumentation schemes, it might then happen that the presented framework, which is built on OWL and SWRL, is not expressive enough to fully capture the original meaning. In this case, it would be necessary to look, for example, to SWRL-FOL (First-Order Logic) for increased expressive power.

Another aspect is improving Protege's functionality. For development with SWRL-FOL, it would be convenient for the Protege editor to support it in addition to normal SWRL. As automatic code generation was also one of the goals kept in mind, improving the Java code generator in Protege is also necessary. Currently, it provides only a class framework without paying attention to class restrictions, SWRL rules etc. In order to create agents from the ontology itself with little additional work, these constraints have to be considered.

Previously, the `truthValue` predicate was introduced, allowing agents to assert that a certain statement is false. However, as the datatype `xsd:boolean` allows only the two values "true" and "false", expressing the fact that a statement is simply unknown to the agent becomes difficult. Omitting the boolean predicate in an argument object might serve this purpose, but this needs to be investigated further.

Finally, the subject of trust has not yet been addressed in this paper. The presented ontology does nothing to prevent agents from lying or not fulfilling their obligations, therefore, other agents would benefit from storing information about past interactions. This is in itself a large field of research and a combination with this ontology of argumentation should prove interesting.

4 Conclusions

Argumentation research has produced a large amount of argumentation models, which are often difficult to compare. In this paper, we have presented a starting point for argumentation research using semantic web technologies, thereby providing a underlying platform from which more sophisticated systems can be built.

The provided ontology can now be extended in various ways, to provide greater power of expression or model additional features such as trust.

References

Rah03. Iyad Rahwan, *Argumentation-based negotiation*, ArgMAS Proceedings, 2003

McG04. Deborah L. McGuinness, World Wide Web Consortium, *OWL Web Ontology Language Overview*, 2004

Kly04. Graham Klyne, World Wide Web Consortium, *Resource Description Framework (RDF): Concepts and Abstract Syntax*, 2004

Hor04. Ian Horrocks, *SWRL: A Semantic Web Rule Language*, 2004

Pat05. Peter F. Patel-Schneider, *A Proposal for a SWRL Extension towards First-Order Logic*, 2005

Ver05. Bart Verheij, Agentlink Technical Forum Group, An Argumentation Core Ontology, 2005

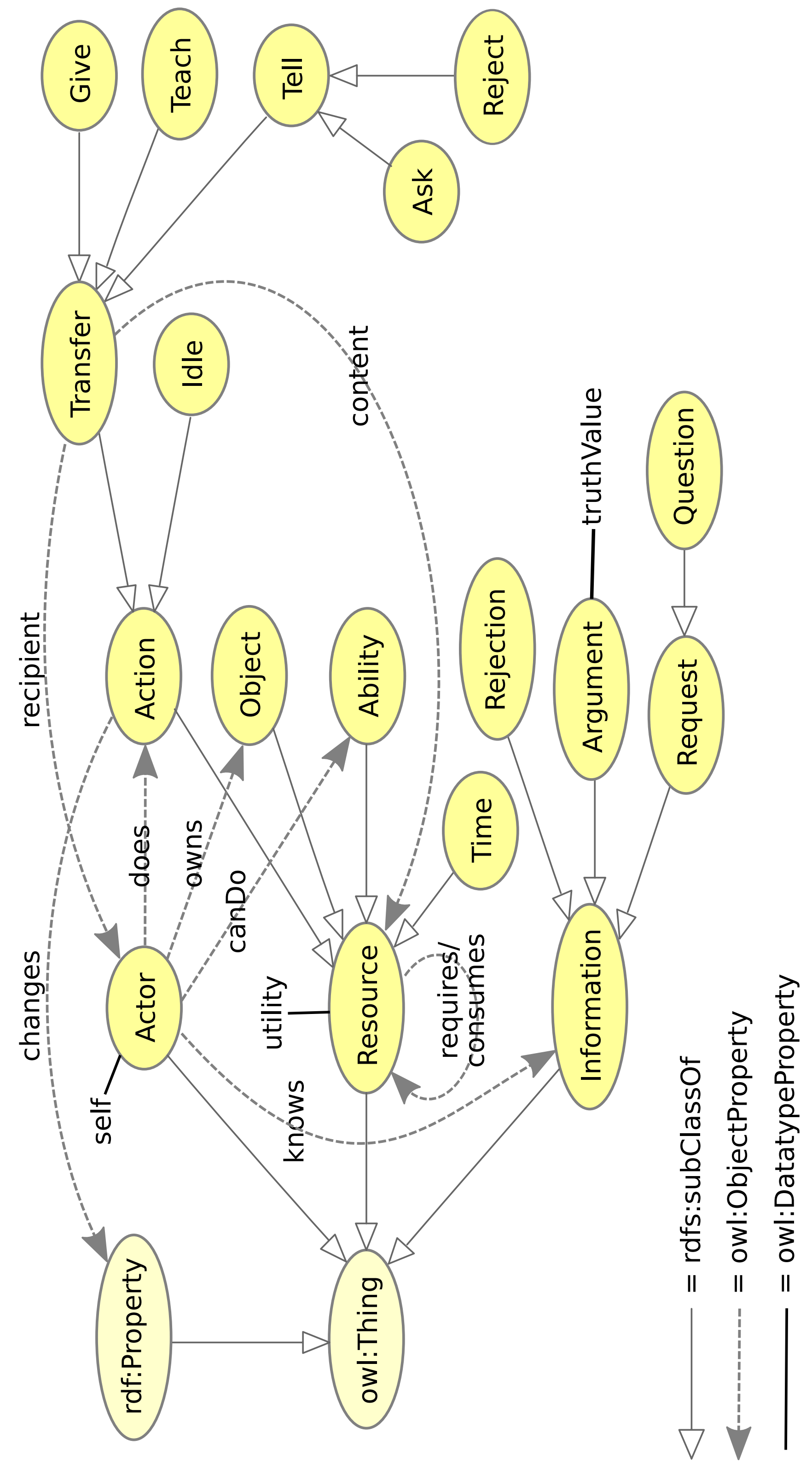

Fig. 1. ontology class relations