SAM : Semantic Agent Model for SWRL rule-based agents

Julien Subercaze, Pierre Maret

SAM
Semantic Agent Model for SWRL rule-based agents

Julien Subercaze
Université de Lyon, LIRIS UMR 5205, INSA de Lyon, Villeurbanne, France
julien.subercaze@liris.cnrs.fr

Pierre Maret
Université de Lyon, LaHC UMR 5516, Université de Saint-Etienne, France
pierre.maret@univ-st-etienne.fr

Keywords: Autonomous agent, agent architecture, rule-based agent, semantic web, behavior exchange, consistency checking, SWRL.

Abstract: Semantic Web technologies are part of multi-agent engineering, especially regarding knowledge base support. Recent advances in the field of logic for the semantic web enable a new range of applications. Among them, programming agents based on semantic rules is a promising field. In this paper we present a semantic agent model that allows SWRL programming of agents. Our approach, based on the extended finite state machine concept, results in a three layers architecture. We detail the architecture, the syntax of the rules , the agent interpreter cycle and present a prototype validating the concept. We present two distinguished features of our approach : behavior exchanges and consistency checking.

1 Introduction and motivation

Since the publishing of the agent roadmap in 2003 (Luck et al., 2003) that pointed out the lack of connection between Multi-Agent Systems and Semantic Web technologies, many applications and frameworks have been developed to bridge this gap. Semantic Web languages and tools are widely used to represent agents' knowledge. TAGA (Zou et al., 2003) uses OWL and RDF as knowledge representation in the field of a trading agent competition, using a FIPA compliant framework. AgentOWL (Laclavik et al., 2006) extends JADE agents with OWL support for their knowledge Base (KB). It also introduces an OWL based semantic agent model. Knowledge Agents, introduced by (Aridor et al., 2000), are used for domain specific web search. In this case, agents KB is based on RDF. RDF is also used in CORESE (Corby et al., 2004) which is a semantic web search engine for corporate knowledge developed within the COMMA (Corporate Memory Management through Agents) european IST project. The JADE framework, which is currently the most used in research and industry supports natively RDF for representing agents' knowledge.

These examples show us that semantic web technologies are widely used for representing agent knowledge. However, agent behaviour programming has difficulties to take advantage of these technologies. We have just identified some emerging proposals in this field. Buhler et al. (Buhler and Vidal, 2003) introduced a language called Picola in which agent behaviour is a composition of Semantic Web Services. More recently, the S-APL (Semantic agent programming language) was introduced by Katasonov (Katasonov and Terziyan, 2008). This language, which is the most advanced attempt of agent semantic programming is built on top of JADE and CWM (Closed World Machine), a rule based reasoning engine. CWM performs first order predicate logic inference. Consequently S-APL doesn't take advantage of the description logic that underpins semantic web technologies. In practical terms, agent knowledge base are represented in RDF and first order predicate logic inferences are performed using CWM. The fact that S-APL is based on CWM means that it is restricted to closed world assumption (Damasio et al., 2006). Closed world assumption implies that everything that is not known to be true, is false. The opposite of closed world assumption is the open world assumption. Open world assumption states that everything that is not known is undefined. As stated in

(Damásio et al., 2006), the incompleteness of knowledge owned by agents is the reason for using the open world assumption in MAS.

Our motivation is to build an agent model that takes advantage of Description Logic expressivity and its reasoning tools. Figure 1 shows the current status of specification in the Semantic Web *layer cake*. The logic part, which is of primary interest for us, is still work in progress. For this layer, two proposals are pending. The most well known one is the Semantic Web Rule Language (SWRL)[1] (Horrocks et al., 2004), it is based on a combination of the OWL DL with the RuleML language. The second proposal is the Web Rule Language (WRL)[2] initiative that was influenced by the Web Service Modeling Language WSML. Whereas WRL is at a draft stage, the semantic web community is focusing its research towards SWRL. Indeed Protege [3], Pellet [4] and Jess [5] already provide support for SWRL even if the reference document is only at the submission stage, and RIF (Rule Interchange Format) [6] which specifies the interoperation with data and ontology languages, is in close spirit with SWRL. Due to these advances in implementation, it is now possible to develop agents based on semantic rules. Thus our choice naturally went to SWRL for the design of the Semantic Agent Model (SAM).

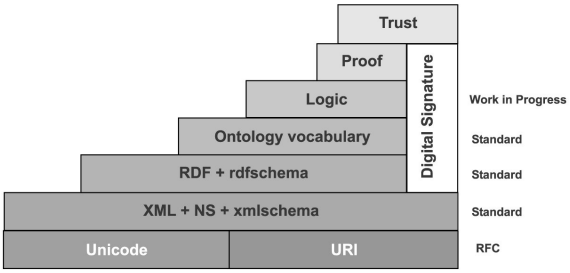

Figure 1: The Semantic Web Layer Cake advances

SWRL presents two main advantages compared to other rule languages. First it allows the writing rules in terms of OWL concepts (i.e. classes, individuals, properties and data values) because it is an OWL based language. To these OWL concepts, the SWRL specification adds several built-ins functions for comparisons, math, strings and time (Horrocks et al., 2004). From a more agent programming point of view, concepts of the agent knowledge base can be directly manipulated in the rule language. Thus, knowledge and behaviors are stored and manipulated at the same level of the agent architecture. Together with the agent model defined in section 3, manipulating knowledge and behaviour at the same level enables behavior exhanges without the overhead of an explicit behaviour description layer, such as in (Decker and Sycara, 1996; Katasonov and Terziyan, 2008). Secondly, the decidable subset of Description Logic (DL) that is used (DL safe rules) in SAM, possesses complete and sound reasoning mechanisms (Baader et al., 2003). We describe in section 4 how these reasoning mechanims are used in SAM for maintaining the consistency of the agent's knowledge base.

In the next section we explain the construction of our agent's model. We first introduce the layered architecture, then detail the control structure and give a formal description of the SAM grammar. In section 3 we describe the ontological model of the agent that results from the architecture. After the description of the agent architecture and its semantic model, we present in section 4 behaviour exchanges and consistency checking features and their applications in Multi-Agent Systems. Our conclusions are presented in section 6.

2 Building Agents with Semantic Rules

2.1 SAM agent Architecture

Programming agent behaviour using a rule language can be carried out in two ways. The first way consists in extending a logic programming language in order to support traditional agent features (i.e. message passing, threading, etc.). The second way consists in building a layered architecture using the rule language at an upper layer. Agent features are delegated to a lower layer. Commonly, in this type of architecture, the lower level language (i.e. Java, C++,etc.) is used to handle communication, file access, thread management, etc. The main idea behind this approach is to reuse the required features for MAS that are already implemented in another language and to define an agent interpreter to support a particular architecture, such as BDI for instance. The literature shows examples of both approaches. Clark et al. (Clark et al., 2001) follows the first approach by extending Qu-Prolog with multi-threading support and inter-thread message communication. However, this approach is not scalable and does not comply with the

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

[2] http://www.w3.org/Submission/WRL/

[3] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab

[4] http://clarkparsia.com/pellet/

[5] http://herzberg.ca.sandia.gov/jess/

[6] http://www.w3.org/TR/rif-core/

Agent Communication Language (ACL) specified by the FIPA [7]. FIPA-ACL is currently recognized as the standard for agent communication and ensures interoperability between MAS frameworks. S-APL, that we discussed in the previous section, follows the same approach but some direct calls to JAVA functions are inserted into the rules.

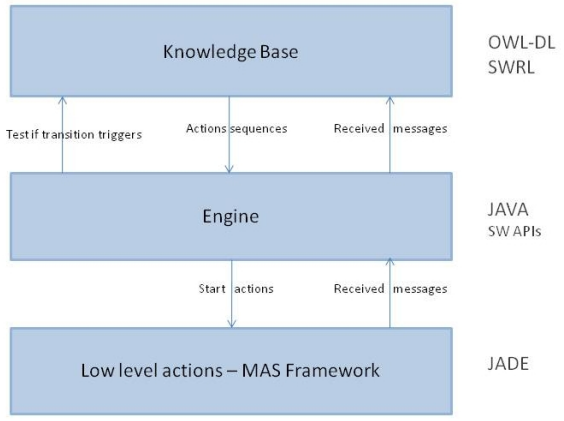

Figure 2: SAM Agent Architecture

Standard MAS languages rely on the second approach. Agent0, the first agent dedicated language, which is an implementation of Shoham's Agent Oriented Programming was developed on top of LISP. Equally, 3APL, 3APL-m, JASON and the BDI agent system Jadex are based on JAVA.

Our architecture follows the second approach and results in the following layered architecture (Fig. 2) :

1. Knowledge Base
2. Engine
3. MAS framework and low level actions

2.1.1 Knowledge Base

The knowledge base is the upper layer of the SAM architecture. The knowledge base contains the knowledge of the agent which, in our approach, is composed of static knowledge and behaviour. Behaviour of agent is expressed using SWRL rules. As SWRL is based upon OWL, terms of the knowledge base are directly manipulated in the rules. Terms of the knowledge base can appear in both antecedent and consequent of rules. A formal specification of the rule syntax is given in section 2.3.

[7] http://www.fipa.org/repository/aclspecs.html

2.1.2 Engine

As SWRL buid-ins do not cover all the requirements for agent programming , we have introduced additional low level actions (3rd layer). This middle layer is the control structure that make the interface between the rules contained in the knowledge base and the low level actions. Rules from the knowledge base are fired by the engine, one at a time. If the rule implies to call low level actions, the engine layer carries out this call.

2.1.3 Low level actions and MAS Framework

This layer contains the implementation of the low level actions that are complementary to SWRL built-ins. An extensive list of these actions is given in section 3. Notice that these actions are introduced as instances of OWL class Actions in the syntax of the rules (1st layer). Communication between agents relies on an existing MAS framework. Messages are structured following the FIPA-ACL standard, consequently the MAS framework has to be FIPA compliant (our implementation is based upon JADE). Messages from other agents are received through the MAS framework, then converted into an OWL representation and finally added to the knowledge base.

2.2 Control structure

Rule-based agents constitutes an important part of the research on MAS. In (Hindriks et al., 1999b), Hindriks et al. define the requirement for a minimal agent programming language that includes rules and goals. They also defined formalization tools that were applied to three standard agent programming languages AGENT-0(Shoham, 1991), AgentSpeak(L)(Rao, 1996) (that was later implemented and extended in JASON(Bordini and Hubner, 2006)) and 3APL(Hindriks et al., 1999a). Their definition of an agent program for goal directed agents includes a set of rules Γ called the rule base of the agent. They identify rule ordering as a crucial issue in rule-based agents. However, this presents us with the following problem : when several rules from the ruleset can be fired, there must be an order to determine the sequence of execution of those rules. So the order in which the rules will be sorted must be defined. Hindriks et al. (Hindriks et al., 1999b) proposed that all rules fall into one of the following categories : *reactive(R)*, *means-end(M)*, *failure(F)* and *optimisation(O)* with an order based on intuition :

$$R > F > M > O$$

As SWRL doesn't support rule ordering, we are also confronted with the same issue. However, instead of deciding an arbitrary order, we have decided to use another model of behavior, a slightly modified version of the Extended Finite State Machine (EFSM) model (Cheng and Krishnakumar, 1993), that guarantees the execution of only one rule at a time. In EFSM, transitions between states are

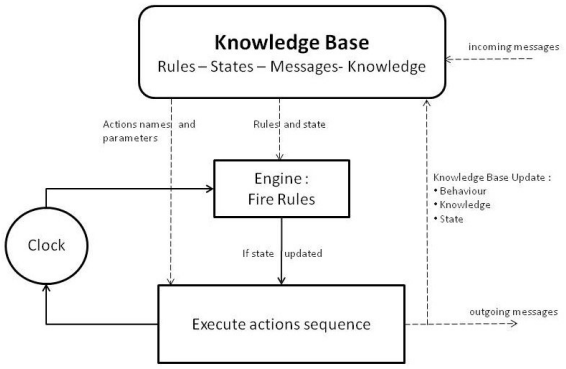

Figure 3: SAM Agent Interpreter

expressed using *if statements*. A transition is fired if trigger conditions are valid. Once the transition has been fired, the machine is brought from current state to next state and a set of specified operations are performed. Our choice is to use atomic actions to fullfill basic MAS requirements. We differentiate two kinds of atomic actions, external and internal. Internal actions have an effect on the agent internal knowledge Base. External actions are the interactions of the agent within its environment. These actions include environment perception, action on the environment, message reception and emission. External actions are not included in SWRL built-ins whereas a subset of internal actions is. In section 3 we detail the list of atomic actions that are not SWRL built-ins. A deterministic EFSM is a restriction of EFSM in which there is at most one possible transition for each state and set of triggering conditions. We used this restriction to ensure that only one rule can be triggered at a time. A pseudo code algorithm for the interpreter is defined in algorithm 1.

Algorithm 1: SAM Interpreter

```
begin
    CurrentState ⟵ sBegin
    while CurrentState ≠ sEND do
        temp ⟵ nextStateValue()
        if temp ≠ currentState then
            removeProperty(currentState, stateValue)
            actionList ⟵ getActionList()
            if executeAction(actionList) then
            addProperty(currentState, temp)
            else
            addProperty(currentState, errorState)
end
```

2.3 Language Syntax

The syntax of the rule language that we designed (given in figure 4) is expressed in Extended Backus-Naur Form (EBNF). This syntax is based on the existing SWRL EBNF syntax as specified in (Horrocks et al., 2004). SAM grammar is a subset of the SWRL grammar. In the antecedent of a SAM rule (*SAMantecedent*) it is mandotary to specify to which state the rule applies. This is set up by the *hasStateValue* property. The previous property, *currentState*, ensures that the rule will be fired when the current state of the EFSM is the one to which the rule applies. The second part of the antecedent contains the triggering conditions. In this part, conditions under which the transition will be triggered are defined. The range of these conditions is the knowledge base of the agent. These conditions are represented by *atom** which is not modified from the original SWRL specification. Conditions can test the validity of class belonging, property between classes or between individuals, including received messages.

The rule consequent term (*SAMconsequent*) specifies the destination state of the transition and the sequence of atomic actions to be executed. Each action has different parameters. Parameters are passed using two properties, *hasParameterName* and *hasParameterValue*. The first property applies to the action which is to be executed and specifies the name of the parameter. Then *hasParameterValue* is applied to the name of the parameter in order to specify its value.

3 Semantic Agent Model

The architecture, control structure and language syntax we have just seen enable us to elaborate the se-

```
SAMrule      ::= 'Implies(' [ URIreference ]
                 { annotation }
                 SAMantecedent SAMconsequent ')'

SAMantecedent::= currentState('i-variable')'
                 hasStateValue'('i-variable')' atom*

SAMconsequent::= hasNextState'('i-variable')'
                 hasActionList'(a-list')' atom*

a-list       ::= hasValue(action) hasNext(a-list)
               | endlist

action       ::= URIreference hasParameterName(a-name)

a-name       ::= hasParameterValue(i-object)

atom         ::= description '(' i-object ')'
               | dataRange '(' d-object ')'
               | individualvaluedPropertyID '
               (' i-object i-object ')'
               | datavaluedPropertyID '
               (' i-object d-object ')'
               | sameAs '(' i-object i-object ')'
               | differentFrom '(' i-object i-object ')'
               | builtIn '(' builtinID { d-object } ')'

builtinID    ::= URIreference

endlist      ::= URIreference

i-object     ::= i-variable | individualID

d-object     ::= d-variable | dataLiteral

i-variable   ::= 'I-variable(' URIreference ')'

d-variable   ::= 'D-variable(' URIreference ')'
```

Figure 4: EBNF interpreted by SAM

mantic agent model. Using the previous given architecture, we built an OWL representation of the agent with different components (Figure 5. The components of which we will now detail. First of all, there is a finite number state, a list of possible atomic actions and the parameters for the actions. We defined two specials states, *sBegin* and *sEnd* that specify the beginning and end states of the EFSM. Every agent's behaviour must start with *sBegin* and end with *sEnd*. Environment interactions are modelized by the received messages queue.

Possible actions that are not SWRL built-ins are divided into two categories : internal and external actions. Here we detail the different atomic actions that we require in both categories.

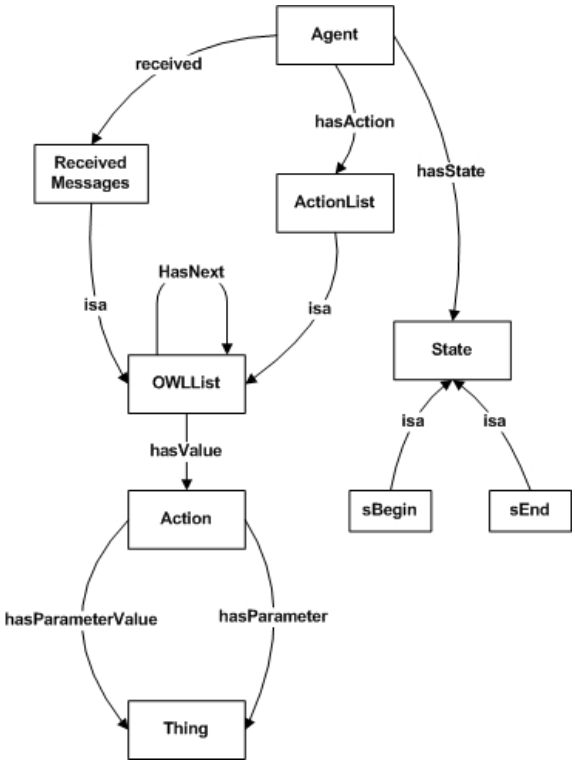

Figure 5: The semantic agent model

Internal Actions : agent knowledge is expressed using OWL concepts : classes, properties, individuals and data value. For each concept, three basic operations are needed : creation, modification, deletion. Unfortunately only the first one is supported by SWRL built in. SWRL supports assertion but does not support negation. In practical terms, it is possible to assert that properties apply to individuals or classes in the rule consequent. The following example is taken from the SWRL proposal document and shows the assertion of the uncle property by composing parent and brother properties :

$$parent(?x,?y) \wedge brother(?y,?z) \Rightarrow uncle(?x,?z) \quad (1)$$

However the following rules (2,3) are not possible since SWRL neither supports negation as a failure (2) nor non-monotonicity (3). Hence it is not possible to withdraw information using the rule consequent.

$$\neg Person(?x) \Rightarrow NonHuman(?x) \quad (2)$$

$$parent(?x,?y) \wedge brother(?y,?z) \Rightarrow \neg aunt(?x,?z) \quad (3)$$

As only creation is possible using SWRL (at a higher level), we define additional actions (modification, deletion) at lower level:

- Modify/remove property
- Modify/remove class belonging from a ressource

- Modify/delete individual
- Modify/delete datarange property

Among internal actions, we made the distinction between SWRL built-ins that are executed by the rule engine and the other required actions that in our model, are the low level atomic actions. These latter are called by the agent interpreter.

External Actions refer to the agents' interactions with their environment. We restrict our scope to software agents that evolve in an electronic environment. Interactions are then limited to message exchanges between agents. We rely on the FIPA ACL specification for the message structures. Received messages are stored in the messagelist. In the agent's KB, messages are put in a list *ReceivedMessages* that is an instance of OWLList[8]. Eventually there as two basic external actions, *sendMessage* and *receiveMessage*. Following the ACL specification, forging a message requires several parameters, among them we can cite sender, receiver, ontology used, performative and so on. From those simple actions, it is possible to build complex interactions between actions, for instance FIPA ACL specifies an extensive communicative act library including query-answer, contracting, proposal, subscribing. Different fields of the message are represented in the OWL knowledge Base using properties, i.e. *hasPerformative*, *hasContent*, *hasSender*

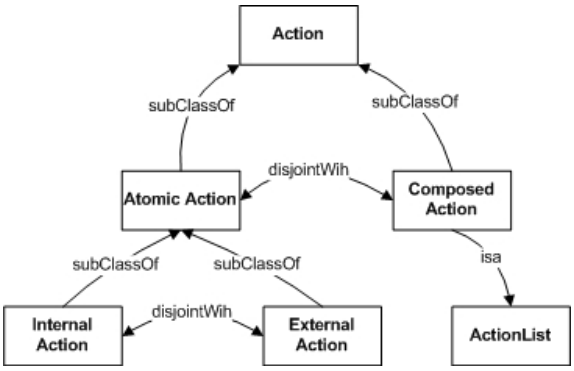

Figure 6: Semantic agent ontology : Actions

3.1 Defining New Actions

The agent model contains a finite list of basic actions for communication and knowledge base management purposes. In SAM there two approaches to define new actions. The first it to extend the set of available low level actions. The second one is to define new actions by combining the existing ones. Defining new atomic actions requires to implement them in low level language. This approach is then of low interoperability and is discouraged by the authors. It should be applied only in case of an extension of the model. The regular approach consists in defining new actions as a sequence of atomic ones. We denoted these actions as composed actions (Fig. 6). Actually, behavior of agents is a kind of composed action since it is composed of a sequence of actions, triggered during a transition. To define new composed actions, we use the same representation as for agent's behaviours. Composed actions are a set of rules that represent an EFSM. These rules should only be active when the composed actions is called. Therefore these rules are not stored as SWRL rules in the knowledge base of the agent but they are instances of the class Rule and their value is a string representation of the rule (In Manchester Syntax [9]). The process of execution of a composed action is the following. Assuming that the agent is firing a transition between state *A* and *B*. During this transition a composed action called *comp* is to be executed. First the engine removes the rules of the current behaviour from the knowledge base and stores them using a string representation. The engine also keeps tracks of the current state and transition sequence that was executed. The engine sets the current state of the agent to an intermediate state *sBegin*. Then it extracts the string representation of the rules from *comp* and add them to the knowledge base. The composed action is then executed following the same way as an agent behaviour. Once the action is finished, the engine removes the rules and set back the agents behaviour context. Note that this process is recursive and a composed action can call another composed action.

4 Features and Benefits

This architecture, as described in previous sections presents two main advantages over existing semantic agent architectures. Firstly, in SAM, behaviors are represented using a rule language that follows the same syntax as the knowledge representation (SWRL and OWL). This way, behaviour and knowledge are represented and stored in the same layer of the architecture. Combining this feature with the fact that agents are aware of their own architecture (Section 3), behaviour exchanges among agents are natively enabled in SAM. Agents can extract their behaviour (or part of their global behaviour) and transmit this latter to other agents as a set of semantic rules.

[8] http://www.co-ode.org/ontologies/lists/

[9] http://www.co-ode.org/resources/reference/manchester_syntax/

Behaviours exchanges as supported in existing framework are made using an extra layer. For instance S-APL uses a Reusable Atomic Behaviour Layer that stores atomic behaviours coded in a non semantic language (i.e. JAVA). In SAM, behaviours are a set of semantic rules and are independent of the implementation of low level actions. Consequently behaviors exchanges can be done between different implementations of the SAM architecture. The interoperability between agents is then increased since the implementation language is no more a problem for reusing behaviors.

Secondly, the logical foundation of SAM agents differs from existing agents framework. Description Logic tools provide reasoning mechanism that allows consistency checking. From the agent point of view, being able to deal with the consistency of the knowledge base is of great interest, on a single agent and on a multi-agent point of view. For a single agent, consistency checking ensures that the received knowledge from external sources is consistent with its internal knowledge. Reasoners such as Pellet or Fact++ allow incremental consistency checking that is used to maintain dynamically the consistency of the agent's knowledge base. In multi-agent argumentation, and especially in the frame of negotiation, consistency checking can be used to provide a great advantage to agents (Parsons et al., 1998). Indeed, being able to prove that the argumentation of the other party is valid or not is a key for successful argumentation. Moreover, DL reasoning tools are not only able to detect inconsistencies but are also able to provide explanations of these inconsistencies. For argumentation, this allows to go one step further, after detecting inconsistencies in the argumentation, it is possible to provide evidence of the argumentation failure.

We implemented examples for both behavior exchanges and consistency checking in the prototype presented in the next section.

5 Implementation

We have developed a JAVA interpreter that communicates with the knowledge Base using the Protege-OWL API [10] Pellet is used in combination with Jena [11] as a OWL and SWRL reasoner. The JADE framework is used for the low level external actions and to provide communication facilities between agents. The framework handles agent registration, service discovery and message passing. It also provides an environment that is FIPA-ACL compliant and thus ensures interoperability with FIPA-ACL compliant frameworks. Since OWL does not support RDF lists, we used OWLList to represent action sequences and for the queue of received messages. The open-source prototype is available online [12].

Along to the validation of the model, the implementation showed us some limitations. We have used Pellet as a SWRL reasoner, since it is currently the most advanced open-source implementation of SWRL. As developments stands at the moment, several important features are not supported by Pellet, for instance some SWRL built-ins are not yet available.

The implementation results show the feasability of the proposal and we intend to further develop the prototype to make it fully suitable for the development of applications.

6 Conclusion and perspectives

In this paper we showed how the next generation of Semantic Web technologies can be applied in MAS programming. We presented an agent model called SAM that enables agent development using the Semantic Web Rule Language (SWRL). We described the three layer architecture, the OWL agent model, its rule syntax and we validated our approach by the implementation of a prototype.

We presented two main features of SAM. We described how behaviours exchanges in SAM can be made regardless of the low level implementation language, making SAM agents "real" semantic agents. Description Logic that underpins and which is inherent in SWRL is a very powerful logic and it allows greater agent reasoning capabilities than standard Prolog. Description Logic tools enable dynamic consistency checking, that is used to maintain agent's knowledge base consistency and to provide explanation of inconsistencies. Further research will focus on the development of applications based on these features, especially in the field of multi-agent argumentation and negotiation.

REFERENCES

Aridor, Y., Carmel, D., Lempel, R., Soffer, A., and Maarek, Y. S. (2000). Knowledge agents on the web. In *CIA '00: Proceedings of the 4th International Workshop on Cooperative Information Agents IV, The Future of Information Agents in Cyberspace*, pages 15–26, London, UK. Springer-Verlag.

[10] http://protege.stanford.edu/plugins/owl/

[11] http://jena.sourceforge.net/

[12] http://code.google.com/p/semanticagent/

Baader, F., Calvanese, D., McGuinness, D., Patel-Schneider, P., and Nardi, D. (2003). *The description logic handbook: theory, implementation, and applications*. Cambridge Univ Pr.

Bordini, R. and Hubner, J. (2006). BDI agent programming in AgentSpeak using Jason. In *Proceedings of 6th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA VI)*, volume 3900, pages 143–164. Springer.

Buhler, P. and Vidal, J. (2003). Semantic web services as agent behaviors. *Agentcities: Challenges in Open Agent Environments*, pages 25–31.

Cheng, K. and Krishnakumar, A. (1993). Automatic functional test generation using the extended finite state machine model. In *Proceedings of the 30th international conference on Design automation*, pages 86–91. ACM New York, NY, USA.

Clark, K., Robinson, P., and Hagen, R. (2001). Multi-threading and message communication in Qu-Prolog. *Theory and Practice of Logic Programming*, 1(03):283–301.

Corby, O., Dieng-Kuntz, R., and Faron-Zucker, C. (2004). Querying the semantic web with corese search engine. In *ECAI*, volume 16, page 705.

Damasio, C., Analyti, A., Antoniou, G., and Wagner, G. (2006). Supporting open and closed world reasoning on the web. *LECTURE NOTES IN COMPUTER SCIENCE*, 4187:149.

Damásio, C. V., Analyti, A., Antoniou, G., and Wagner, G. (2006). Open and closed world reasoning in the semantic web. In *Proceedings of IPMU 2006), special session Works on the Semantic Web*, pages 1850–1857, Paris, France. Editions E.D.K. Participaçãoo por convite e sujeita a avaliação.

Decker, K. and Sycara, K. (1996). Designing reusable behaviors for information agents. Technical report.

Hindriks, K., De Boer, F., Van der Hoek, W., and Meyer, J. (1999a). Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401.

Hindriks, K., De Boer, F., Van Der Hoek, W., and Meyer, J. (1999b). Control structures of rule-based agent languages. In *Atal'98: Paris, France*, page 384. Springer.

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member Submission*, 21.

Katasonov, A. and Terziyan, V. (2008). Semantic agent programming language (S-APL): A middleware platform for the Semantic web. In *Proc. 2nd IEEE International Conference on Semantic Computing*, pages 504–511.

Laclavik, M., Balogh, Z., Babik, M., and Hluchý, L. (2006). Agentowl: Semantic knowledge model and agent architecture. *Computers and Artificial Intelligence*, 25(5).

Luck, M., McBurney, P., and Preist, C. (2003). *Agent technology: Enabling next generation computing*. AgentLink II.

Parsons, S., Sierra, C., and Jennings, N. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292.

Rao, A. (1996). AgentSpeak (L): BDI agents speak out in a logical computable language. *Lecture Notes in Computer Science*, 1038:42–55.

Shoham, Y. (1991). AGENT0: A simple agent language and its interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, volume 2, pages 704–709.

Zou, Y., Finin, T., Ding, L., Chen, H., and Pan, R. (2003). Using semantic web technology in multi-agent systems: a case study in the taga trading agent environment. In *ICEC '03: Proceedings of the 5th international conference on Electronic commerce*, pages 95–101, New York, NY, USA. ACM.