SAM : Semantic Agent Model for SWRL rules 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
Saint-Etienne, France
pierre.maret@univ-st-etienne.fr

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 an 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.

1. Introduction and motivation

Since the publishing of the agent roadmap in 2003 [14] 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 [19] uses OWL and RDF as Knowledge representation in the field of a trading agent competition, using a FIPA compliant framework. AgentOWL [13] extends JADE agents with OWL support for their Knowledge Base (KB). It also introduces an OWL based agent semantic model. Knowledge Agents, introduced by [1], are used for domain specific web search. In this case, agents KB is based on RDF. RDF is also used in CORESE [6] 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.

While using RDF or OWL to represent agents' knowledge is a common, programming agents' behaviour with Semantic Web technologies is rare. We identified emerging proposals. [3] introduced a language called Picola in order to define behaviours as a composition of Semantic Web Services. More recently, the S-APL (Semantic agent programming language) was introduced by Katasonov [12]. 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 inference. Consequently S-APL doesn't take advantage of the Description Logic that stands behind semantic web technologies. Practically, agent's KB are represented in RDF and FOPL inferences are performed using CWM. S-APL is based on RDF, therefore it imposes straight monotonic construct and is restricted to closed world assumption [8]. As stated in [7], the incompleteness of knowledge owned by the agents is the motivation for using non-monotonic reasoning and open world assumption. Non-monotonic reasoning imply that adding a rule to

Our motivation is to build an agent model that takes advantage of Description Logic expressivity, open world reasoning and nonmonotonic reasoning. On a more technical point of view, we also aim at defining a model relying on a semantic rule language and that can be implemented in several lower language such as JAVA, C++. We base our approach on Semantic Web (SW) advances. Latest advances in Semantic Web development enable the development of new agent programming language. Figure 1 shows current status of specification in the Semantic Web layer cake. The logic part, which is of primary interest for us, is still a work in progress. For this layer, two proposals are pending. The most well known is the Semantic Web Rule Language (SWRL)[1] [11] one, it is based on a combination of the OWL DL and OWL lite with the RuleML language. The second is the Web Rule Language (WRL)[2] initiative that was influenced by the Web Service Modeling Language WSML. Whereas WRL is at a draft step, the Semantic Web community drives its research towards SWRL. Indeed Protege, Pellet and Jess already provides support for SWRL even if the reference document is only at the submission step. Thanks to this 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). SWRL presents two main advantages compared to other rule languages. First it is an

1. http://www.w3.org/Submission/SWRL/
2. http://www.w3.org/Submission/WRL/

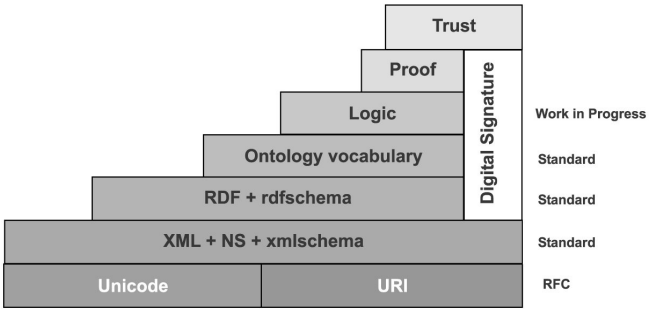

Figure 1. The Semantic Web Layer Cake - SW rules langages

OWL-based rule language, consequently it allows to write rules in terms of OWL concepts, i.e. classes, individuals, properties and data values. To these OWL concepts, the SWRL specification adds several built-ins functions for comparisons, math, strings and time [11]. On the agent programming point of view, it allows to manipulate concepts of the agent's Knowledge Base directly in the rule language.

The second benefit is the logical foundation of SWRL. It combines OWL-DL (decidable version of OWL) with Rule Markup Language (RuleML). SWRL can be roughly considered as the union of Horn-Logic and OWL based on the description logic $\mathcal{SHOIN}$. Consequently the expressivity of SWRL comes at the price of decidability [16]. SWRL is not decidable. But it exists a subset called *DL Safe SWRL rules* that is decidable. For the agent development, DL Safe SWRL is more expressive compared to other rule languages. . Most of the rule-based agents are based on Prolog supporting Horn Clauses. Researchs are currently under development for implementing DL reasoning in Prolog but none is currently available for agents. Practical advantages of using DL in the field of Multi-Agent Systems (MAS) has been shown in [15], especially in the field of information retrieval. We previously stressed on the benefit of non-monotonic reasoning. It is currently not supported in SWRL, but a restriction of SWRL rules allows extension to enable non-monotonic reasoning [11].

In the next section we detail the construction of our agent's model. We first introduce the layered architecture, then detail the control structure,give an EBNF of the SAM grammar. In section 3 we describe the ontological model of the agent that results from the architecture. Section 4 shows a practical example of a SAM behaviour and details the different steps of its execution. Section 5 concerns the implementation of the SAM prototype. Our conclusions are presented in section 6.

2. Building Agents with Semantic Rules

2.1. 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. [5] 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 Agent Communication Language (ACL) specified by the FIPA [3]. 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.

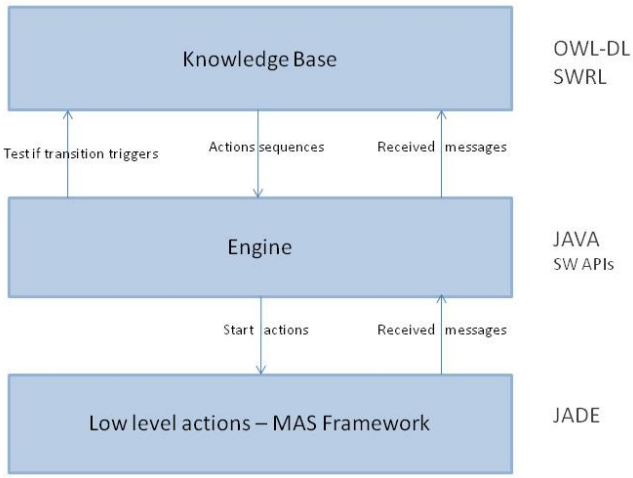

Figure 2. SAM Agent Architecture

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.

3. http://www.fipa.org/repository/aclspecs.html

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

1) Knowledge Base
2) Engine
3) Low level actions

2.1.1. Knowledge Base.

2.1.2. Engine.

2.1.3. Low level actions. A REFORMULER The semantic layer contains the agent Knowledge Base, the SWRL rules and the OWL model of agent that we will discuss later. In this layer all the components are based on OWL and rules are written in SWRL. The rule engine executes the extended finite state machine built in SWRL. When a transition fires it retrieves the sequence of actions to execute and their parameters. The engine part is made of a SWRL rule engine that fires rules and an interpreter to chain the transitions and start low level actions with their respective parameters.

2.2. Control structure

Rule-based agents constitutes an important part of the research on MAS. In [10], 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[18], AgentSpeak(L)[17] (that was later implemented and extended in JASON[2]) and 3APL[9]. 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. [10] 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 [4], that guarantees the execution of only one rule at a time. In EFSM, transitions between states are 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

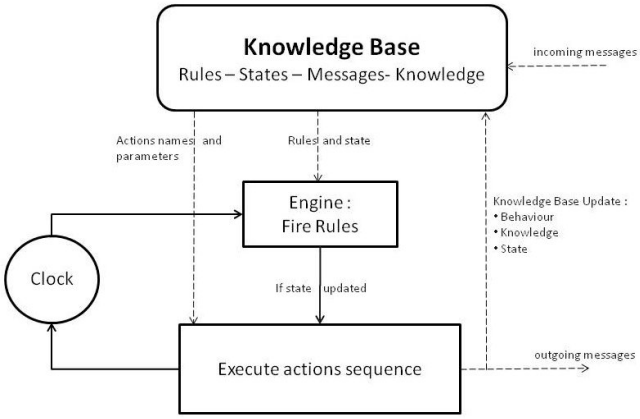

Figure 3. SAM Agent Interpreter

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 **??** we detail the list of non SWRL built-in atomic actions. 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 2.3) is expressed in Extended Backus-Naur Form (EBNF). This syntax is based on the existing SWRL EBNF syntax as specified in [11]. SAM grammar is included in the

```
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

SWRL grammar.

$$SAM\ Grammar \subset SWRL\ Grammar$$

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 semantic 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.

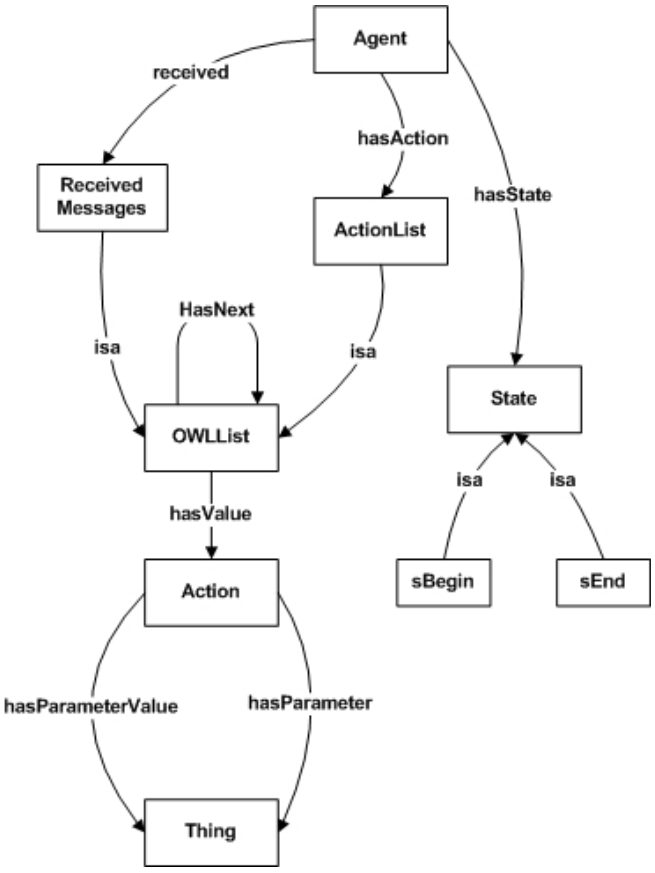

Figure 5. The semantic agent model

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.

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, we defined the internal actions that are not supported by the rule language :

- 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-in functions 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 individual from Class. OWLList[4]. 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*

3.1. Defining New Actions

The agent model contains a finite list of basic actions for communication and knowledge base management purpose. In SAM there a two approaches to define new actions. The first it to extend the set of available of low level actions. The second one is to define new actions by combining the existing ones. Defining new atomic actions require 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 by a sequence of actions, triggered by transition. To define new composed actions, we use the same representation as for agents 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). 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 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. It is in fact similar to the calls stack principle in computer hardware.

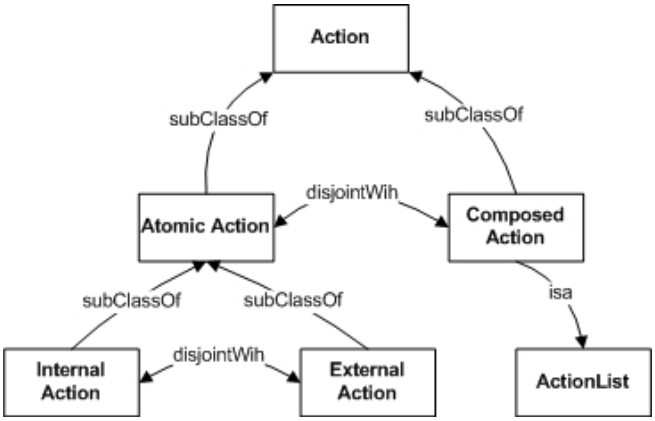

Figure 6. Semantic agent ontology : Actions

4. http://www.co-ode.org/ontologies/lists/

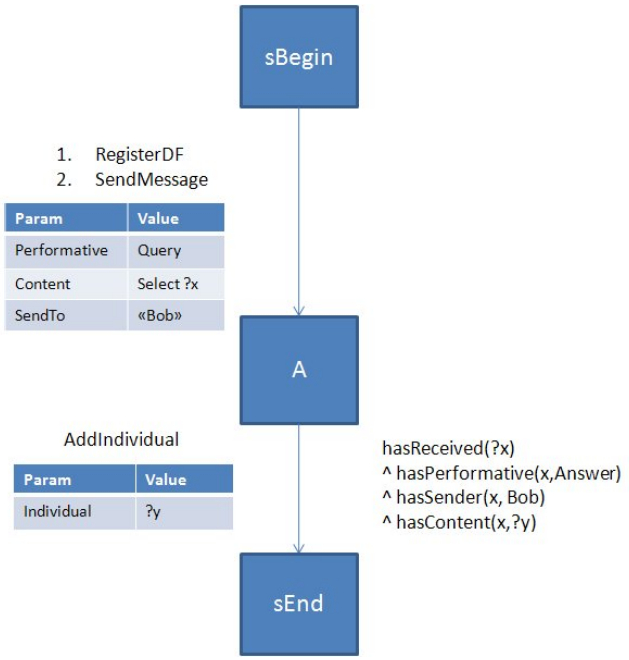

Figure 7. Illustrative example (a), Scenario

4. Example

To illustrate the mechanism behind semantic agents, we will take a simple example and process the several steps of the execution. Following example will start the agent *Alice*, register with the directory facilitator of the framework, send a query to the agent *Bob*. If the received message is from *Bob* and is this message has the performatrive answer then *Alice* adds the content of the answer into its knowledge base. The resulting EFSM is depicted in figure 7. The left column of the figure describes the low level atomic actions executed during a transition. Triggering conditions, that are in the antecedent of the rule, are on the right side of the picture. The first transition is conditions free. If *Alice* is in the *sBegin* state then the transition to the state A will occur. Ations are executed as a sequence. The next actions is executed only if the previous succeeded. First *registerDF* will be executed. If it returns true the instruction a message with the query performative, containing a query will be sent to the agent *Bob*. The rule used to describe this transition is presented below in a human readable syntax :

$$
\begin{aligned}
& CurrentState(?x) \\
& \wedge\, hasStateValue(x, sBegin) \\
& \wedge\, NextState(?y) \\
& \\
& \Rightarrow\, hasStateValue(y, A) \\
& \wedge\, hasContents(ActionSequence, registerDF) \\
& \wedge\, hasNext(ActionSequence, item) \\
& \wedge\, hasContents(item, SendMessage) \\
& \wedge\, hasParameterName(SendMessage, Sender) \\
& \wedge\, hasParameterValue(SendTo, Bob) \\
& \ldots\, same\ with\ other\ parameters \\
& \wedge\, hasNext(item, endList)
\end{aligned}
$$

Within the architecture, the engine checks whether a transition occurs by asking the knowledge base the *NextState* value. If this value is different from the *CurrentState* one then a transition is enabled. Engine then retrieves the values of *ActionSequence*, with the respectives parameters. *ActionSequence* is actually a simply linked-list (Fig.8) in which each item has as the *hasParameterName* property. The name of the parameter of the value has the *hasParameterValue* property to specify the value of the parameter. The structure of the list of actions follows the OWL model depicted in figure 5. For this transition, only the *Sendmessage* instructions has parameters. The instruction is linked to its parameters using properties as described in figure 8.

The second transition contains triggering regarding the received message. As *Alice* sent a query to *Bob*, the next step of the behaviour is to handle the answer from *Bob*. Thus, we specify condition on the received message to ensure that *Bob* is the sender and that the message is of type *Answer*.

$$
\begin{aligned}
& CurrentState(?x) \\
& \wedge\, hasStateValue(x, A) \\
& \wedge\, NextState(?y) \\
& \wedge\, hasReceived(?z) \\
& \wedge\, hasPerformative(z, Answer) \\
& \wedge\, hasSender(z, Bob) \\
& \wedge\, hasContent(z, ?w) \\
& \\
& \Rightarrow\, hasStateValue(y, sEnd) \\
& \wedge\, hasContents(ActionSequence, AddInvidual) \\
& \wedge\, hasNext(ActionSequence, endList) \\
& \wedge\, hasParameterName(AddInvidual, name) \\
& \wedge\, hasParameterValue(name, w)
\end{aligned}
$$

We will then detail the interactions between the different layers in the architecture during the execution of the first transition.

4.1. Execution Phase

Representing the action execution following the architecture as in Section 2.1 on a timeline is represented in Figure 9. It follows Algorithm 1. The SAM engine firstly enquires of a rule triggering, in this case, the Knowledge Base query returns $NextState = A$. As $A \neq sBegin$, the engine retrieves the current list of actions containing *RegisterDF* and *SendMessage*. Actions are performed sequentially. First *RegisterDF* is executed and if it returns true then *SendMessage* will be executed. When both actions succeeded, the current state of the agent is updated to *NextState* value, in this case it is A.

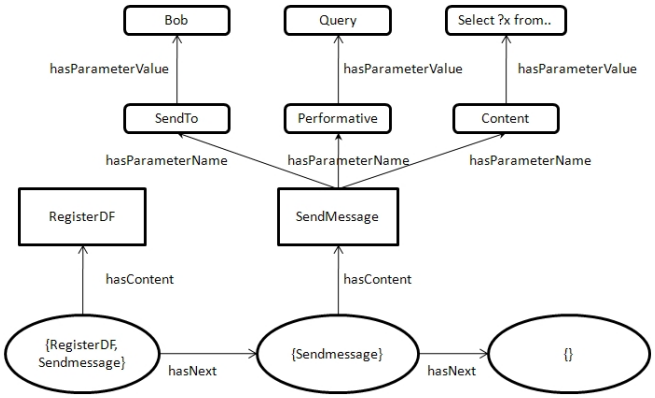

Figure 8. Illustrative example (b), ActionList data structure

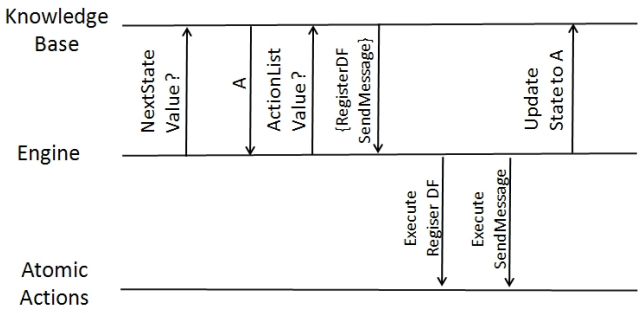

Figure 9. Illustrative example (c), Flow chart of the transition from Begin to A

5. Implementation

We developed a JAVA interpreter that communicates with the Knowledge Base using the Protege-OWL API. Pellet is used in combination with Jena as a SWRL reasoner. The JADE framework is used for the low level external actions. The framework handles agent registration, service discovery and messages passing. It also provides an environment that is FIPA-ACL compliant. One implementation issue we encountered is that OWL does not support RDF lists. Fortunately, an OWL equivalent called OWLList has been developed and is used to represent action sequences and the queue of received messages. A first version of the open-source prototype is available online [5].

Along to the validation of the model, the implementation showed us some current limitations. Nowadays status of SWRL reasoners is not satisfying because none of them fully support the SWRL specification. We used Pellet as SWRL reasoner, since it is currently the most advanced open-source implementation. At the current stand of development, several main features are not supported by Pellet, for instance SWRL built-ins are not yet available.

The implementation results shows the feasability of the proposal and we intend to further develop the prototype to make it fully suitable for the development of applications. Advances in the field of the Semantic Web technologies being very fast, current restrictions on SWRL supports should soon belong to the past and allow further development of the prototype. Finally, this implementation of the prototype allowed us to validate our approach and to identify the limitations.

6. Conclusion and perspectives

In this paper we showed how next generation of Semantic Web technologies can be applied in MAS programming. We presented an agent model called SAM that allows to develop agent using the Semantic Web Rule Language. We described an extensive model with a detailed architecture, a semantic agent model, its rules syntax and validated our approach by the implementation of a prototype.

Description Logic that stands behind SWRL is a very powerful logic and allows greater agent reasoning capabilities than standard Prolog. It will allow in a near future the development of powerful Knowledge Management and Semantic Web applications relying on multi-agent systems.

In current MAS applications based on Semantic Web technologies, agents behaviour is hardcoded in a lower level language making behaviour exchanges a complex task for agents. In SAM, integration of the behaviours as part of the agents knowledge base (SWRL is OWL compatible) combined with the reflexivity of the agent model enable native behavior exchanges. As behaviour exchanges are related to learning processes, this feature is of very interesting purpose for cognitive agents applications.

Our contribution is situated at a lower level of an agent programming language. SAM is then a step forward an assembly language for semantic agents. Several complex architectures, as for instance BDI, can be implemented on top of it. Further research will focus on the improvement of the model and the prototype, the development of an IDE. We are also investigating possible relationships with the Agent UML modeling language.

5. http://code.google.com/p/semanticagent/

References

[1] Yariv Aridor, David Carmel, Ronny Lempel, Aya Soffer, and Yoëlle S. Maarek. 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, 2000. Springer-Verlag.

[2] R.H. Bordini and J.F. Hubner. 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.

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

[4] K.T. Cheng and AS Krishnakumar. 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, 1993.

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

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

[7] Carlos Viegas Damásio, Anastasia Analyti, Grigoris Antoniou, and Gerd Wagner. Open and closed world reasoning in the semantic web. In *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-06), special session Works on the Semantic Web*, pages 1850–1857, Paris, France, July 2006. Editions E.D.K. Participaçãoo por convite e sujeita a avaliação.

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

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

[10] K.V. Hindriks, F.S. De Boer, W. Van Der Hoek, and J.J.C. Meyer. Control structures of rule-based agent languages. In *Intelligent Agents V: Agent Theories, Architectures, and Languages: 5th International Workshop, Atal'98: Paris, France, July 1998: Proceedings*, page 384. Springer, 1999.

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

[12] A. Katasonov and V. Terziyan. 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, 2008.

[13] Michal Laclavik, Zoltan Balogh, Marian Babik, and Ladislav Hluchý. Agentowl: Semantic knowledge model and agent architecture. *Computers and Artificial Intelligence*, 25(5), 2006.

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

[15] Ralf Mller, Volker Haarslev, and Bernd Neumann. Expressive description logics for agent-based information retrieval. In *Treur (Eds.), Knowledge Engineering and Agent Technology, IOS*. Press, 2000.

[16] B. Parsia, E. Sirin, B.C. Grau, E. Ruckhaus, and D. Hewlett. Cautiously Approaching SWRL. Technical report, Technical report, University of Maryland, 2005.

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

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

[19] Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan. 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, 2003. ACM.