Verifying Resource Requirements for Ontology-Driven Rule-Based Agents

Conference Paper · March 2012

DOI: 10.1007/978-3-642-28472-4_18

CITATIONS
6

READS
84

3 authors:

Abdur Rakib
University of the West of England, Bristol
49 PUBLICATIONS **220** CITATIONS
SEE PROFILE

Rokan Uddin Faruqui
McMaster University
7 PUBLICATIONS **41** CITATIONS
SEE PROFILE

Wendy Maccaull
St. Francis Xavier University
93 PUBLICATIONS **602** CITATIONS
SEE PROFILE

Verifying resource requirements for ontology driven rule-based agents

Abdur Rakib, Rokan Uddin Faruqui, and Wendy MacCaull

StFX Centre for Logic and Information
St. Francis Xavier University, Canada
{arakib, x2010mcd, wmaccaul}@stfx.ca

Abstract. Recent efforts towards the Semantic Web have resulted in powerful languages such as Semantic Web Rule Language (SWRL) based on OWL-DL and RuleML. Rule languages and inference engines incorporate reasoning capabilities to Semantic Web application systems. In this paper we present an approach for the design and specification of ontology-driven multi-agent rule-based systems. We use the Maude rewriting system and its Linear Temporal Logic (LTL) model checking tool to verify response time guarantees for the target systems. We present TOVRBA, an extended version of a verification tool developed by the first author, for ontology-driven multi-agent rule-based systems which allows the designer to specify information about agents' interactions, behavior, and execution strategies at different levels of abstraction. TOVRBA generates an encoding of the system for the Maude LTL model checker, allowing properties of the system to be verified. We illustrate the use of the framework on a simple healthcare system.

Keywords: Multi-agent systems, Semantic Web, Ontology, Model checking.

1 Introduction

There has recently been considerable interest in Semantic Web and rule-based approaches to various aspects of agent technology. The integration of the Semantic Web and intelligent agents research has been realized [11, 35], and intelligent agents are considered as a promising approach towards realizing the Semantic Web vision [20]. In order to make better use of the artificial intelligent research experience and the practical application of Semantic Web technologies, the World Wide Web Consortium (W3C) has adopted the Web ontology language (OWL) as a language for processing Web information [8]. While the adoption of the standard ontology language OWL provides the basis for some forms of reasoning, it has been realized that rules are a key element of the Semantic Web vision [22]. For example, ontology based reasoning using DAML/OWL and, more generally, in rule extensions to ontologies (OWL extended with Horn-style rules) such as ORL [22] and SWRL [23] significantly increase the expressive power of ontology languages [22]. While rule-based systems are rapidly becoming an important

component of Semantic Web application, the resulting system behavior and the resources required to realize them, namely, how to ensure the correctness of rule-based designs (will a rule-based system produce the correct output for all legal inputs), termination (will a rule-based system produce an output at all) and response time (how much computation will a rule-based system have to do before it generates an output) can be difficult to predict. These problems become even more challenging for distributed rule-based systems, where the system being designed or analyzed consists of several communicating rule-based programs which exchange information via messages. A communicated fact may be added asynchronously to the state of a rule-based system while the system is running, potentially triggering a new strand of computation which executes in parallel with current processing. In order to provide response time guarantees for such systems, we must know how much time a rule-based system needs to perform the required reasoning. Furthermore, for a rule-based system running on a PDA or other mobile device, the number of messages exchanged may be a critical factor.

In this paper, we present a verification framework for an ontology-driven system that supports automated verification of time and communication requirements in distributed rule-based agents. We consider distributed problem-solving in systems of communicating rule-based agents, and ask how much time (measured as the number of rule firings) and how many message exchanges it takes the system to find a solution. We use standard model checking techniques to verify interesting properties of such systems, and show how the Maude model checker can be used to verify properties including response-time guarantees of the form: if the system receives a query, then a response will be produced within n time steps.

The remainder of the paper is organized as follows. In section 2 we provide an overview of ontologies and how rules are derived from ontologies which are the basis of designing our rule-based agents. In section 3 we describe our model of communicating ontology-driven rule-based agents. An example illustrating the approach is provided in section 4. In section 5 we briefly describe a tool TOVRBA for translating ontology based specification of the agents into Maude, and in section 6 we present some experimental results using TOVRBA. We discuss related work in section 7 and conclude in section 8.

2 Ontology-driven Horn clause rules

Ontologies and rules play a central role in the design and development of Semantic Web applications. In this section, first we briefly discuss ontologies and their integration with rules. Then we show how we extract Horn clause rules from ontologies to design our rule-based agents.

2.1 Ontologies and rules

An ontology is an explicit formal specification of a conceptualization which defines certain terms of a domain and the relationships among them [18]. The Web

ontology language OWL is a semantic markup language for ontologies that provides a formal syntax and semantics for them. The W3C declared two different standardizations for OWL: OWL 1 [27] and OWL 2 [28]. The first standardization has three profiles, namely OWL Lite, OWL DL, and OWL Full. The OWL Lite profile is less expressive than the other two profiles and the OWL DL profile is based on description logic (DL). Description logic based OWL is a good candidate for defining ontologies where automated reasoning is required to exploit reasoning algorithms. The OWL Full profile has the maximal expressive power but reasoning over an ontology using all features of OWL Full is undecidable [27]. For efficient and tractable reasoning, a new standardization was declared called OWL 2, which trades some expressive power for more efficient reasoning [26]. OWL 2 DL has three sublanguages known as profiles, namely, OWL 2 EL, OWL 2 QL, and OWL 2 RL; each of them is useful in different application scenarios. All the profiles exhibit a polynomial time complexity for ontological reasoning tasks. For the rest of this paper we refer OWL 2 DL as OWL 2.

In this work, we use the OWL 2 RL and SWRL [23] languages for defining ontologies and rules. OWL 2 RL is suitable for the design and development of rule-based systems. The inspiration behind the design of OWL 2 RL is pD^* [24] and description logic programs (DLP) [17]. OWL 2 RL describes the domain of an ontology in terms of classes, properties, individuals, and datatypes and values. Individual names refer to elements of the domain, e.g., `Mary`; classes describe sets of individuals having similar characteristics, e.g., `Patient`; properties describe binary relationships between pairs of individuals, e.g., `isFeeling` describes a relationship between two individuals `Mary` and `MucositisPainTwo`. Here `Mary` is an individual of the class `Patient` and `MucositisPainTwo` is an individual of the class `Pain`. Notice that properties may be object or data properties depending on the types of individuals. Similarly, datatypes represent types of literal values (e.g., integers). An object property links an individual to an individual, whereas a datatype property links an individual to a data value. In OWL 2 RL, object properties can be functional, inverse functional, reflexive, irreflexive, symmetric, asymmetric, and transitive; however, data properties can be only functional [28]. In addition, complex class descriptions can be constructed using all the above components (classes, properties, individuals, and datatypes and values) and various constructors including union and intersection.

In OWL 2, International Resource Identifiers (IRIs) are used to identify ontologies and their elements. OWL 2 has several syntaxes, including the Manchester Syntax [3] and the Functional-Style Syntax [4]. In this paper, we use the Functional-Style Syntax and we consider the Unique Name Assumption (UNA) for each element of our ontology; the latter allows us to omit the prefix IRI during the translation of an ontology to derive a rule-based system. To illustrate, we present OWL 2 RL axioms (1)-(4) for an ontology about pain management. In order to define the elements of the ontology, we declare a prefix IRI a as `<http://logic.stfx.ca/ontologies/PainOntology.owl#>`. The first axiom defines the concept "Either a doctor or a nurse or a relative of a patient can be a caregiver". The second and third axioms assert the facts that "Mary is a

patient" and "Mary is feeling Mucositis Pain". The last axiom characterizes the object property `isFeeling` as the inverse object property of `isFeltBy`.

$$Prefix(a :=< http://logic.stfx.ca/ontologies/PainOntology.owl\# >)$$

$$SubClassOf(ObjectUnionOf(DoctorNurseRelative)CareGiver) \quad (1)$$

$$ClassAssertion(a : Patient a : Mary) \quad (2)$$

$$ObjectPropertyAssertion(a : isFeeling a : Mary a : MucositisPainTwo) \quad (3)$$

$$InverseObjectProperties(a : isFeeling a : isFeltBy) \quad (4)$$

Both the description logic based OWL 1 and OWL 2 are decidable fragments of first order logic; however, the expressive power of OWL 1 is strictly limited to certain tree structure-like axioms [17]. For instance, a simple rule $hasFather(?x, ?y) \wedge hasBrother(?y, ?z) \rightarrow hasUncle(?x, ?z)$ can not be modeled using OWL 1 axioms. Although OWL 2 can express this *uncle* rule indirectly, many rules are still not possible to model using OWL 2 axioms [25]. Function-free Horn clause rules can remove such restrictions while being decidable but they are restricted to universal quantification and no negation [29]. A combination of OWL 2 with rules offers a more expressive formalism for building Semantic Web applications. Several proposals have been made to combine rules with ontologies. We use one of them, the SWRL that extends OWL DL by adding new axioms, namely Horn clause rules. Although SWRL was a proposed extension for OWL 1, it can be used as an rule extension for OWL 2 [16]. We combine a set of SWRL rules with the set of OWL 2 RL axioms and facts to build our ontology. For example, the statement "if a patient has pain level "2" then the pain intensity (s)he is feeling is mild" can be written in SWRL rule format in Functional-Style syntax as follows:

$Prefix(var :=< urn : swrl\# >)$
$DLSafeRule($
$Body($
$ClassAtom(a : PatientVariable(var : p))$
$ObjectPropertyAtom(a : isFeelingVariable(var : p)Variable(var : x))$
$ClassAtom(a : PainVariable(var : x))$
$DataPropertyAtom(a : hasPainLevelVariable(var : x)$"2"^^$xsd : string)$
$)$
$Head($
$ObjectPropertyAtom(a : hasPainIntensityVariable(var : p)a : MildPainIn)$
$))$

More precisely, the SWRL rule defined above states that an individual p from the `Patient` class who is feeling pain x that has pain level "2" asserts the fact that p has pain intensity `MildPainIn`. Here `MildPainIn` is an individual of the class `MildPain`. Such rule is best described in plain text syntax as follows:

$$Patient(?p) \wedge isFeeling(?p, ?x) \wedge Pain(?x) \wedge hasPainLevel(?x, "2") \rightarrow hasPaintIntensity(?p, MildPainIn) \quad (5)$$

2.2 Translation of ontologies into rules

Since OWL 2 RL is based on DLP, the set of axioms and facts of an OWL 2 RL ontology can be translated to Horn clause rules [17]. In order to design an ontology-driven rule-based system, first we use the DLP framework [17] to translate an ontology to a set of Horn clause rules. In OWL 2 RL, facts are described using ClassAssertion and ObjectPropertyAssertion/DataPropertyAssertion which correspond to DL axioms of the form $a : C$ and $\langle a, b\rangle : P$, respectively, where a and b are individuals, C is a class, and P is an object/data property. Note that these facts are already in the Horn clause rule format with empty bodies. For instance, the facts in (2) and (3) are translated as `Patient(Mary)` and `isFeeling(Mary,MucositisPainTwo)`.

OWL 2 Axioms and Facts	DL Syntax	Horn clause rule
`ClassAssertions`	a:C	C(a)
`PropertyAssertion`	$\langle a, b\rangle : P$	$P(a, b)$
`SubClassOf`	$C \sqsubseteq D$	$C(x) \rightarrow D(x)$
`EquivalentClasses`	$C \equiv D$	$C(x) \rightarrow D(x)$,$D(x) \leftarrow C(x)$
`EquivalentProperties`	$P \equiv Q$	$Q(x, y) \rightarrow P(x, y)$
		$P(x, y) \rightarrow Q(x, y)$
`ObjectInverseOf`	$P \equiv Q^-$	$P(x, y) \rightarrow Q(y, x)$
		$Q(y, x) \rightarrow P(x, y)$
`TransitiveObjectProperty`	$P^+ \sqsubseteq P$	$P(x, y) \wedge P(y, z) \rightarrow P(x, z)$
`SymmetricObjectProperty`	$P \equiv P^-$	$P(x, y) \rightarrow P(y, x)$
`Object/DataUnionOf`	$C_1 \sqcup C_2 \sqsubseteq D$	$C_1(x) \rightarrow D(x)$, $C_2(x) \rightarrow D(x)$
`Object/DataIntersectionOf`	$C \sqsubseteq D_1 \sqcap D_2$	$C(x) \rightarrow D_1(x)$,$C(x) \rightarrow D_2(x)$
`Object/DataSomeValuesFrom`	$\exists P.C \sqsubseteq D$	$P(x, y) \wedge C(y) \rightarrow D(x)$
`Object/DataAllValuesFrom`	$C \sqsubseteq \forall P.D$	$C(x) \wedge P(x, y) \rightarrow D(y)$
`Object/DataPropertyDomain`	$\top \sqsubseteq \forall P^-.C$	$P(y, x) \rightarrow C(y)$
`Object/DataPropertyRange`	$\top \sqsubseteq \forall P.C$	$P(x, y) \rightarrow C(y)$

Table 1. Translation of OWL 2 RL axioms and facts into Horn clause rules

The syntax of OWL 2 RL is asymmetric, i.e., the syntactic restrictions allowed for subclass expressions differ from those allowed for superclass expressions. For instance, an existential quantification to a class expression (`ObjectSomeValuesFrom`) is allowed only in subclass expressions whereas universal quantification to a class expression (`ObjectAllValuesFrom`) is allowed only in superclass expressions. These restrictions facilitate the translation of OWL 2 RL axioms into Horn clause rules based on the DLP framework. Translations of some of the OWL 2 RL axioms and facts into rules are given in Table 1. In the second column, complete DL statements are given which are constructed by the corresponding OWL 2 RL axioms and facts to illustrate the translation. For example, `ObjectIntersectionOf`($\sqcap$) is represented by the statement $C_1 \sqsubseteq D_1 \sqcap D_2$. The translation of SWRL rules is straightforward because they are already in the Horn clause rule format. In our approach, agents in a multi-agent rule-based system are designed using the translated Horn clause rules of an ontology. The translation process is automated and is a part of the Tovrba tool (cf. § 5).

3 Systems of communicating rule-based agents

We adopt the model of distributed agents presented in [6]. A distributed reasoning system consists of n_{Ag} (≥ 1) individual reasoners or *agents*. Each agent is identified by a value in $\{1, 2, \ldots, n_{Ag}\}$ and we use variables i and j over $\{1, 2, \ldots, n_{Ag}\}$ to refer to agents. An agent in the system is either concrete or abstract. Each concrete agent has a program, consisting of Horn clause rules (derived from OWL 2 RL + SWRL), and a working memory, which contains facts (ground atomic formulae) representing the initial state of the system. The introduction of ontology-driven rules increases the expressiveness of the framework in [6], and makes it easier to model complex real world Semantic Web problems. In addition, existing tools, including Protégé [5], support the design of OWL 2 RL + SWRL based ontologies, making it easier to model rule-based agents using semantic rules. The behavior of each abstract agent is represented in terms of a set of temporal doxastic formulae. A doxastic logic is a modal logic concerned with reasoning about beliefs. That is, abstract specifications are given as LTL formulae which describe the external behavior of agents, and allow their temporal behavior (the response time behavior of the agent), to be compactly modeled. The agents (concrete and abstract) execute synchronously. We assume that each agent executes in a separate process and that agents communicate via message passing. We further assume that each agent can communicate with multiple agents in the system at the same time.

3.1 Concrete agents

The two main components of rule-based agents are the knowledge base (KB) which contains a set of IF-THEN rules and the working memory (WM) which contains a set of facts that constitute the current (local) state of the system. Another component of a rule-based system is the inference engine which reasons over rules when the application is executed. The inference engine may have some reasoning strategies to handle cases when multiple rule instances are eligible to fire.

In Listing 1.1, we specify the abstract syntax for concrete agents rules using a BNF. In this notation, the terminals are quoted, the non-terminals are not quoted, alternatives are separated by vertical bars, and components that can occur zero or more times are enclosed braces followed by a superscript asterisk symbol ($\{\ldots\}^*$). A class atom represented by `description(i-object)` in the BNF consists of an OWL 2-named class and a single argument representing an OWL 2 individual, for example an atom `Person(x)` holds if `x` is an instance of the class description `Person`. Similarly, an individual property atom represented by `individualvaluedProperty(i-object,i-object)` consists of an OWL 2 object property and two arguments representing OWL 2 individuals, for example an atom `hasCarer(x,y)` holds if `x` is related to `y` by property `hasCarer` and so on. Note that OWL 2 is limited to unary and binary predicates and it is function-free. Therefore, in the Protégé editor all the arguments of *Ask* and *Tell* are represented using constant symbols and these annotated symbols are

translated appropriately when designing the target system using the Maude specification.

```
Rule ::= '<' Priority ':' Atoms '→' Atom '>'
Atoms ::= Atom {∧ Atom}*
Atom ::= standardAtom | commmunicationAtom
standardAtom ::= description'('i-object ')'
            | individualvaluedProperty'('i-object ',' i-object ')'
            | datavaluedProperty'('i-object ',' d-object ')'
            | sameIndividuals'('i-object ',' i-object ')'
            | differentIndividuals'('i-object ',' i-object ')'
            | dataRange'(' d-object ')'
            | builtIn'(' builtinId ',' {d-object}* ')'
communicationAtom ::= 'Ask(' i ',' j ',' standardAtom ')'
            | 'Tell(' i ',' j ',' standardAtom ')'
Priority ::= N≥0
N≥0 ::= 0 | 1 | 2 | ...
i ::= 1 | 2 | ... | nAg
j ::= 1 | 2 | ... | nAg
builtinID ::= URIreference
i-object ::= i-variable | individualID
d-object ::= d-variable | dataLiteral
i-variable ::= 'I-variable('URIreference')'
d-variable ::= 'D-variable('URIreference')'
```

Listing 1.1. Abstract syntax for concrete agents rules

Rules derived from OWL 2 RL + SWRL are translated using TOVRBA into the simplified text format for the encoding and verification of the target system. In other words, the rules of a concrete agent have the plain text format $\langle\ n\ :\ P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow P\ \rangle$ where n is a constant that represents the annotated priority of the rule and the P_i's and P are atoms. For communication, we assume a simple query-response scheme based on asynchronous message passing. Each agent's rules may contain two distinguished communication atoms: $Ask(i, j, P)$, and $Tell(i, j, P)$, where i and j are agents and P is an atomic formula not containing an Ask or a $Tell$. $Ask(i, j, P)$ means 'i asks j whether P is the case' and $Tell(i, j, P)$ means 'i tells j that P' ($i \neq j$). The positions in which the Ask and $Tell$ primitives may appear in a rule depends on which agent's program the rule belongs to. Agent i may have an Ask or a $Tell$ with arguments (i, j, P) in the consequent of a rule; for example, $\langle\ n\ :\ P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow\ Ask(i, j, P)\ \rangle$ whereas agent j may have an Ask or a $Tell$ with arguments (i, j, P) in the antecedent of the rule; for example, $\langle\ n\ :\ Tell(i, j, P) \rightarrow P\ \rangle$ is a well-formed rule (we call it trust rule) for agent j that causes it to believe i when i informs it that P is the case. No other occurrences of Ask or $Tell$ are allowed. When a rule has either an Ask or a $Tell$ as its consequent, we call it a communication rule. All other rules are known as deduction rules. These include rules with Asks and $Tell$s in the antecedent as well as rules containing neither an Ask nor a $Tell$.

Firing a communication rule instance with the consequent $Ask(i, j, P)$ adds the atom $Ask(i, j, P)$ both to the working memory of i and of j. Intuitively, i has a record that it asked j whether P is the case, and j has a record of being asked by i whether P is the case. Similarly, if the consequent of a communication rule instance is of the form $Tell(i, j, P)$, then the corresponding atom $Tell(i, j, P)$ is

added to the working memories of both the agents i and j. The set of facts are ground atomic formulae.

We assume that each concrete agent has a *reasoning strategy* (or conflict resolution strategy) which determines the order in which rules are applied when more than one rule matches the contents of the agent's working memory. The framework (and the TOVRBA tool presented in §5) supports a set of standard conflict resolution strategies often used in rule-based systems [12, 15, 37] including: rule ordering, depth, breadth, simplicity, and complexity. Different agents in the system may use different types of reasoning strategies. To allow the implementation of reasoning strategies, each atom is associated with a time stamp which records the cycle at which the atom was added to the working memory. The internal configurations of the rules follow the syntax given below:

$$\langle\ n\ :\ [\ t_1\ :\ P_1] \wedge [\ t_2\ :\ P_2] \wedge \ldots \wedge [\ t_n\ :\ P_n] \rightarrow [\ t\ :\ P]\ \rangle$$

where the t_i's and t represent time stamps of atoms. When a rule instance of the above rule is fired, its consequent atom P will be added to the working memory with time stamp $t = t' + 1$, i.e., t will be replaced by $t' + 1$, where t' is the current cycle time of the system.

3.2 Abstract agents

An abstract agent consists of a working memory and a behavioral specification. The behavior of abstract agents is specified using the temporal logic LTL extended with belief operators. The decision regarding which agents to abstract and how their external behavior should be specified rests with the system designer. Specifications of the external (observable) behavior of abstract agents may be derived from, e.g., assumed characteristics of as-yet-unimplemented parts of the system, or from assumptions regarding the behavior of parts of the overall system the designer does not control (e.g., quality of service guarantees offered by an existing web service) or from the prior verification of the behavior of other (concrete) agents in the system. The general form of the formulae used to represent the external behavior of an abstract agent i is given below:

$$\begin{array}{lcl}
\rho & ::= & X^{\leq n}\varphi_1 \mid G(\varphi_2 \rightarrow X^{\leq n}\ \varphi_3) \\
\varphi_1 & ::= & B_i\ \mathit{Ask}(i,j,P) \mid B_i\ \mathit{Tell}(i,j,P) \\
 & & \mid B_i\ \mathit{Ask}(j,i,P) \mid B_i\ \mathit{Tell}(j,i,P) \\
 & & \mid B_i\ P \\
\varphi_2 & ::= & B_i\ \mathit{Ask}(j,i,P) \mid B_i\ \mathit{Tell}(j,i,P) \\
\varphi_3 & ::= & B_i\ \mathit{Tell}(i,j,P) \mid\ B_i\ \mathit{Tell}(i,k,P) \\
 & & \mid B_i\ \mathit{Ask}(i,j,P) \mid\ B_i\ \mathit{Ask}(i,k,P)
\end{array}$$

where X is the next step temporal operator, $X^{\leq n}$ is a sequence of n X operators, G is the temporal 'in all future states' operator, and B_i for each agent i is a syntactic doxastic operator used to specify agent i's 'beliefs', i.e., the contents of its working memory. Formulae of the form $X^{\leq n}\varphi_1$ describe agents which produce a certain message or input to the system within n time steps. A formula φ_1 of the form $B_i\ \mathit{Ask}(i,j,P)$ or $B_i\ \mathit{Tell}(i,j,P)$ results in communication with the other agent as follows: when the beliefs appear (as an *Ask* or a *Tell*) in the abstract

agent i's working memory, they are also copied to agent j's working memory at the next step. A formula of the form $B_i\ P$ representing a belief involving an atom P (other than Ask and $Tell$) may also appear in the abstract agent i's working memory within n time steps. This is not critical to how abstract agents interact with communication; however it describes agent i's own behavior.

The $G(\varphi_2 \rightarrow X^{\leq n}\ \varphi_3)$ formulae describe agents which are always guaranteed to reply to a request for information within n time steps. We interpret the formula $G(B_i\ Ask(j,i,P) \rightarrow X^{\leq n}\ B_i\ Tell(i,j,P))$ as follows: if t is the time stamp when abstract agent i came to believe formula $Ask(j,i,P)$ (agent j asked for P), then the formula $Tell(i,j,P)$ must appear in the working memory of agent i within $t+n$ steps. The formula $Tell(i,j,P)$ is then copied to agent j's working memory at the next step. The other possible combinations of Ask and $Tell$ in places of φ_2 and φ_3 in the $G(\varphi_2 \rightarrow X^{\leq n}\ \varphi_3)$ formulae can be interpreted in a similar way. Note that we do not need the full language of LTL (for example, the Until operator) in order to specify these abstract agents, and the language described above for abstract agents is independent of the language of concrete agents.

4 Example of two communicating agents

To illustrate the use of the proposed framework, let us consider an example system consisting of two agents where one is concrete and the other is abstract. We consider a scenario in which agent 1, the concrete agent, has the following set of rules:

$$Rule1 < 1 : prescribedOpioid(?p,' NonOpioidRegimen) \rightarrow Ask(1,2,needAssessment(?p)) >$$

$$Rule2 < 2 : Tell(2,1,isFeeling(?p,?x)) \rightarrow isFeeling(?p,?x) >$$

$$Rule3 < 3 : Patient(?p) \wedge isFeeling(?p,?x) \wedge Pain(?x) \wedge hasPainLevel(?x,' 2) \rightarrow hasPaintIntensity(?p,' MildPainIn)$$

$$Rule4 < 4 : hasPaintIntensity(?p,' MildPainIn) \rightarrow prescribedOpioid(?p,' WeakOpioidRegimen) >$$

For the rest of this paper constants are preceded by a single quote (note that ' and ′ have the same meaning). The first rule states that if a patient is currently in the non-opioid regimen then ask the abstract agent 2 for an assessment. The second rule is a trust rule for agent 1 which makes it trust agent 2 when agent 2 informs it, say for example, `isFeeling('Mary,'MucositisPainTwo)` which is a ground instantiation of `isFeeling(?p, ?x)`. The third rule states that if a patient has a pain level "2" then the pain intensity (s)he is feeling is mild. The fourth rule state that if a patient feels mild pain then the prescribed opioid regimen is weak. The external behavior of the abstract agent 2 is described by the following temporal logic formula:

$$G(B_2\ Ask(1,2,needAssessment('Mary)) \rightarrow X^{\leq 3} B_2\ Tell(2,1,isFeeling('Mary,' MucositisPainTwo)))$$

Time	Agent 1	Agent 2
0	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)]}	{ }
	Rule1	Idle
1	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))]}	{ }
	Idle	Copy (Ask(1,2,needAssessment('Mary)) from Agent 1)
2	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))]}	{[2 : Ask(1,2,needAssessment('Mary))] }
	Idle	Idle
3	{[0 : Pain('MucositisPainTwo)] [0:Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))]}	{[2 : Ask(1,2,needAssessment('Mary))] }
	Idle	Idle
4	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))]}	{[2 : Ask(1,2,needAssessment('Mary))] }
	Idle	$Tell$
5	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))]}	{[2 : Ask(1,2,needAssessment('Mary))] } [5 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))] }
	Copy ($Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo)) from Agent 2)	Idle
6	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))] [6 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))]}	{[2 : Ask(1,2,needAssessment('Mary))] } [5 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))] }
	Rule2	Idle
7	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))] [6 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))] [7 : isFeeling('Mary, 'MucositisPainTwo)]}	{[2 : Ask(1,2,needAssessment('Mary))] } [5 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))] }
	Rule3	Idle
8	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))] [6 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))] [7 : isFeeling('Mary, 'MucositisPainTwo)] [8 : hasPaintIntensity('Mary,'MildPainIn)]}	{[2 : Ask(1,2,needAssessment('Mary))] } [5 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))] }
	Rule4	Idle
9	{[0 : Pain('MucositisPainTwo)] [0 : Patient('Mary)] [0 : prescribedOpioid('Mary, 'NonOpioidRegimen)] [0 : hasPainLevel('MucositisPainTwo,'2)] [1 : Ask(1,2,needAssessment('Mary))] [6 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))] [7 : isFeeling('Mary, 'MucositisPainTwo)] [8 : hasPaintIntensity('Mary,'MildPainIn)] [9 : prescribedOpioid('Mary, 'WeakOpioidRegimen)]}	{[2 : Ask(1,2,needAssessment('Mary))] } [5 : $Tell$(2,1,isFeeling('Mary, 'MucositisPainTwo))] }

Table 2. Example derivation of two communicating agents

which states that whenever the atom *Ask*(1,2,needAssessment('Mary)) appears in the working memory of agent 2 (i.e., agent 2 believes the atom *Ask*(1,2, needAssessment('Mary))), the atom *Tell*(2,1,isFeeling('Mary,'MucositisPainTwo)) will appear in the working memory of agent 2 (i.e., agent 2 will believe atom *Tell*(2,1,isFeeling('Mary,'MucositisPainTwo))) within 3 time steps. Suppose now that the initial working memories of the agents as follows:

WM_1 :{ [0: prescribedOpioid('Mary,'NonOpioidRegimen)] [0 : Patient('Mary)]
[0: hasPainLevel('MucositisPainTwo,'2)] [0 : Pain('MucositisPainTwo)]}
WM_2 :{ }.

Table 2 gives a simple example of a run of the system starting from the initial configuration. This example helps to explain how facts are derived and communicated, and what happens when the abstract agent receives an *Ask* query by communication. Note that in this derivation it is assumed that at step 2, when abstract agent 2 came to believe atom *Ask*(1,2,needAssessment('Mary)) (i.e., agent 1 asked for needAssessment('Mary)), the atom *Tell*(2,1,isFeeling('Mary,'MucositisPainTwo)) appeared in the working memory of agent 2 at time $2 + 3$ i.e., at the 5th step. However, the atom *Tell*(2,1,isFeeling('Mary, 'MucositisPainTwo)) could also appear at the 3rd or 4th step but definitely appears at step 5 if it is not already present in the working memory of agent 2. In the example run, at the 2^{nd} and 3^{rd} steps both agents perform an Idle action. In this model, communication requires a single time step, i.e., when agent i asks a query (or tells an information to) agent j at time step t, agent j will receive the query (information) at time step $t+1$. The table shows that from the initial configuration agents can derive prescribedOpioid('Mary, 'WeakOpioidRegimen) (appears in the configuration of one of the agents) in 9 time steps and exchange two messages.

5 Automated verification tool Tovrba

We use the Protégé version 4.1 ontology editor and knowledge-base framework to build the ontologies and define the rules for concrete agents. The SWRL editor is integrated with Protégé and permits the interactive editing of SWRL rules. In order to encode an ontology-driven rule-based system using a Maude [10] specification and formally verify its interesting properties, we first need to translate the ontology in the OWL/XML format to a set of simple plain text Horn clause rules. We developed a translator that takes as input an OWL 2 RL ontology in the OWL/XML format (an output file of the Protégé editor) and translates it to a set of plain text Horn clause rules. We use the OWL API [21] to parse the ontology and extract the set of axioms and facts. The design of the OWL API is directly based on the OWL 2 Structural Specification and it treats an ontology as a set of axioms and facts which are read using the visitor design pattern. The DLP-based translation rules (cf. § 2.2) are then recursively applied to generate equivalent plain text Horn clause rules for each axiom and fact. We also extract the set of SWRL rules using the OWL API which are already in the Horn clause rule format. First, atoms with corresponding arguments associated

with the head and the body of a rule are identified and we then generate a plain text Horn clause rule for each SWRL rule using these atoms.

The translated Horn clause rules of an ontology are then used to create agents of a multi-agent rule-based system using the Maude specification. We then automatically verify interesting properties of the system using the Maude LTL model checker. The high-level architecture of the TOVRBA tool is shown in Figure 1. We use the Maude rewriting system because it allows efficient modeling of the agents' first order rules and reasoning strategies. For example, the variables that appear in a rule can be represented directly in the Maude encoding, without having to generate all ground instances resulting from possible variable substitutions.

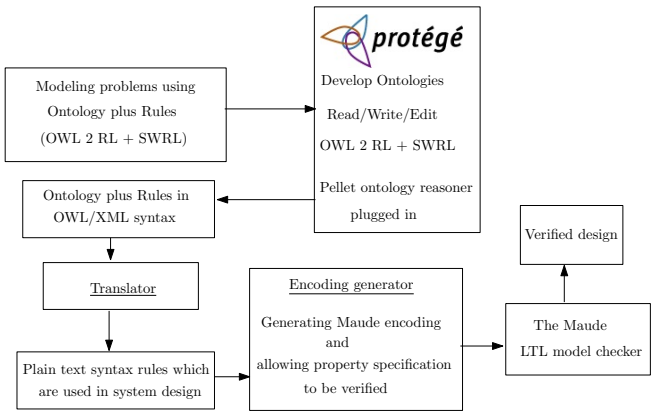

Fig. 1. The TOVRBA tool architecture

In [7] a preliminary description of the tool was published for rule-based multi-agent systems which allows the designer to specify information about agents interactions, behavior, and execution strategies at different levels of abstraction. The extended version of the tool presented in this paper allows the designer to model ontology-driven rule-based agents consisting of rules derived from OWL 2 RL + SWRL. Each agent in a multi-agent system has a configuration (local state) and the composition of all these configurations (local states) make the configuration (global state) of the multi-agent system. The types necessary to implement the local state of an agent (working memory, program, agenda, reasoning strategies, message counters, time stamps etc.) are declared in a generic agent configuration functional module called `ACM`. The structure of the `ACM` is given in Listing 1.2.

```
1  fmod ACM is
2    protecting NAT .
3    protecting BOOL .
4    protecting QID .
5    sorts Constant Atom sAtom cAtom Term Rule Agenda WM .
6    sorts TimeA TimeWM RepT RepTime Config .
```

```
7     subsort Atom < WM .
8     subsorts aAtom cAtom < Atom .
9     subsort Rule < Agenda .
10    subsort Qid < Constant .
11    subsort TimeA < TimeWM .
12    subsorts Constant < Term .
13    subsort RepT < RepTime .
14    ops void rule : -> Atom .
15    ops com exec : -> Phase [ctor] .
16    op nil : -> Term[ctor] .
17    op [_ : _] : Nat Atom -> TimeA .
18    op _ _ : WM WM -> WM [comm assoc] .
19    op _ _ : TimeWM TimeWM -> TimeWM [comm assoc] .
20    op _ _ : Agenda Agenda -> Agenda [comm assoc] .
21    op <_ : _->_> : Nat TimeWM TimeA -> Rule .
22    op _ _ : RepTime RepTime -> RepTime [comm assoc] .
23    op Ask : Nat Nat sAtom -> cAtom .
24    op Tell : Nat Nat sAtom -> cAtom .
25      .
26      .
27      .
28  endfm
```

Listing 1.2. Sorts declaration and their relationships

A number of Maude library modules such as `NAT`, `BOOL`, and `QID` have been imported into the `ACM` functional module. The modules `NAT` and `BOOL` are used to define natural and Boolean values, respectively, whereas the module `QID` is used to define the set of constant symbols (constant terms of the rule-based system). The set of variable symbols (variable terms of the rule-based system) are simply Maude variables of sort `QID`. Both variables and constants are subsorts of sort `Term`. Similarly, an atom is declared as an operator whose arguments are of sort `Term`, and returns an element of sort `Atom`. For example, `op isFeeling : Term Term -> Atom`. The sort `Atom` is declared as a subsort of the sort `WM` (working memory), and a concatenation operator is declared on sort `WM` which is the double underscore shown below.

```
op _ _ : WM WM -> WM [comm assoc] .
```

The above operation is in mixfix notation and it is commutative and associative. This means that working memory elements are a set of atoms whose order does not matter. In order to maintain time stamps for each atom, a sort `TimeA` is declared whose elements are of the form `[ t : P ]`, where `t` represents the time stamp of atom `P` indicating when that atom was added to the working memory. The sort `TimeA` is declared as a subsort of the sort `TimeWM`, and a concatenation operator is declared on sort `TimeWM` which is also the double underscore and is commutative and associative.

```
op _ _ : TimeWM TimeWM -> TimeWM [comm assoc] .
```

Note that `WM` and `TimeWM` are updated simultaneously, for example, whenever an atom `P` is added to `WM` the corresponding element `[ t : P ]` is also added to `TimeWM` for an appropriate time step `t`. Fact time stamps are maintained to implement reasoning strategies. The rules of each agent are defined using an operator which takes as arguments a sort `Nat` specifying the priority, a set of atoms (of sort `TimeWM`) specifying the antecedents of the rule and a single atom

(of sort `TimeA`) specifying the consequent, and returns an element of sort `Rule` (line 21 of Listing 1.2). The sort `Rule` is declared as a subsort of the sort `Agenda`, and a concatenation operator is declared on sort `Agenda` which is also the double underscore and is commutative and associative.

```
op _ _ : Agenda Agenda -> Agenda [comm assoc] .
```

In a similar fashion, other sorts and operators are declared and manipulated. We model each (concrete and abstract) agent using a functional module which imports the `ACM` module defined above. The local configuration of an agent i is represented as a tuple `Si[A|RL|TM|M|RT|RT'|t|msg|syn]iS`, where `Si` and `iS` indicate start and end of a state of agent i. The variables `A` and `RL` are of sort `Agenda`, `TM` is of sort `TimeWM`, `M` is of sort `WM`, `RT` and `RT'` are of sort `RepTime`. Moreover, `t`, `msg`, and `syn` are of sort `Nat`. The variables `t`, `msg`, and `syn` have been used to represent respectively the time step, message counter, and a flag for synchronization. Note that the structure of local configurations for both concrete and abstract agents are the same. This is to maintain consistency of the shape of each agent's configuration. However, for example, the sort `RepTime` is of no use for concrete agents and its value is always empty for them. For the sake of brevity, we do not describe the encoding in any further details here, we refer the interested reader to [32].

6 Pain management system

In this section, we consider an example system adopted from the guidelines for the management of cancer related pain in adults [9]. A simplified flowchart of the guidelines is shown in Figure 2. The guidelines say that if a patient is not currently on a regular opioid regimen then depending on the pain intensity of a patient (s)he will be placed on one of the three regimens non-opioid, weak opioid, or strong opioid. Depending on the patient's response the opioid may be changed, e.g., from non-opioid to weak opioid or weak opioid to strong opioid. The guidelines for the strong opioid regimen say that if a patient is responding (i.e., if the current pain level is equal or less than previous pain level) then another reassessment should be done at an appropriate interval, say within a week. However, if a patient is not responding then it suggests a different reassessment interval depending on the current pain level and considers increasing the dose by a certain percentage. The reassement also suggests that if there are side effects (side effects are symptoms which may occur at any dose e.g., constipation) then management of side effects from opioid therapy will take effect. Also, when on the strong opioid regimen, if a pain crisis occurs with any patient at any point in time, it is an emergency situation.

We built a pain monitoring ontology, that integrates the terminology and concepts of health and medicine used in the Guysborough Antigonish Strait Health Authority (GASHA). A set of standard terms were also obtained from SNOMED-CT [2], ICNP [19], and the guidelines for Cancer pain treatment. We followed the Basic Formal Ontology [1] which is a philosophically inspired top level ontology

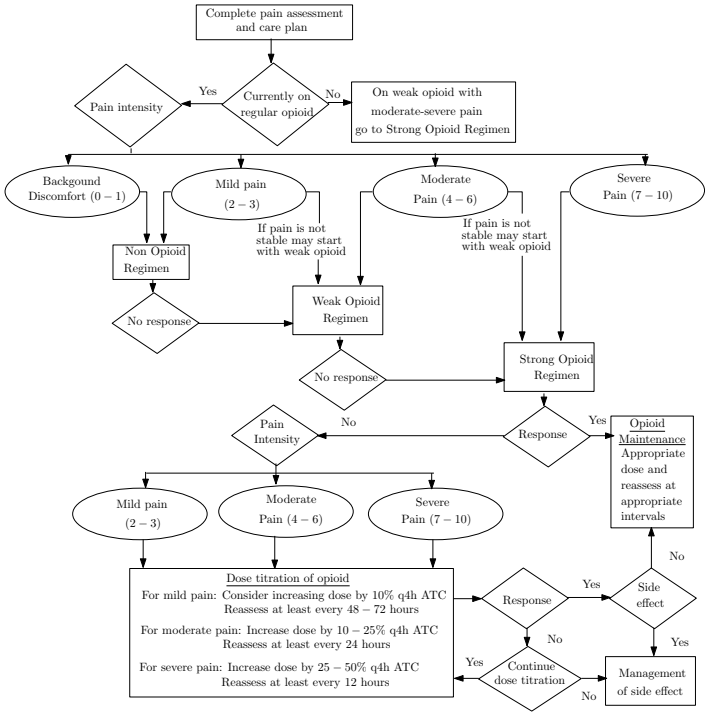

Fig. 2. Cancer pain treatment guidelines [9]

that provides a coherent, unified understanding of the whole domain. A fragment of the pain monitoring ontology is depicted in Figure 3. In our ontology, we have several classes including `Pain`, `Person`, `Patient`, `PainintensityType`, `SpecialPainProblem`, `SideEffects`, object properties include, e.g., `hasPainIntensity, Domain:Pain, Range:PainIntensityType`, and data properties include, e.g., `hasPainLevel, Domain:Pain, Range:xsd:int`. It also includes inverse object properties such as, for example, `isFeeling` and `isFeltBy` and functional object properties, e.g., `hasPainLevel`, i.e., each pain level belongs to an instance of `Pain` class. We also use propositional connectives to create complex class expressions; an example of such expressions is given in axiom 1 (cf. § 2.1).

We model a multi-agent rule-based system using TOVRBA based on the above defined pain monitoring ontology. The system consists of one concrete agent and several abstract agents. The concrete agent in the system is modeled as a central Health Planner $planner(p)$. We model five abstract agents, namely, an assessor $assessor\ (a)$, a reassessor $reassessor(r)$, a side effect manager $sideeffectmanager(s)$, a care giver $caregiver(c)$, and an emergency $emergency(e)$. These abstract agents interact with the agent p and they communicate via message passing. For simplicity, we assume that agent a can be a patient or a family member, agent r

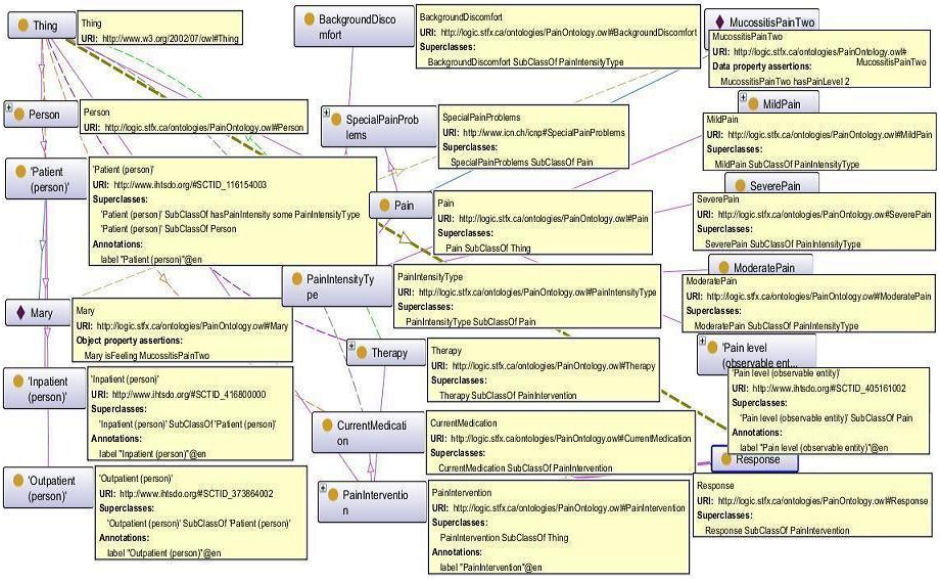

Fig. 3. A fragment of the pain monitoring ontology

can be a physician or a nurse or a relative or a patient herself, and agent c can be a physician or a nurse.

Example rules for the concrete agent p are:

$$< 2 : Patient(?p) \wedge isFeeling(?p, ?x) \wedge Pain(?x) \wedge hasPainLevel(?x,' 2) \rightarrow hasPaintIntensity(?p,' MildPainIn) >$$

$$< 2 : Patient(?p) \wedge isFeeling(?p, ?x) \wedge Pain(?x) \wedge hasPainLevel(?x,' 1) \rightarrow hasPaintIntensity(?p,' BackgroundDiscomfortIn) >$$

The abstract agent a assesses the current pain level of a patient and informs the planner p, with messages of the form: $Tell(a, p, isFeeling(?p, ?x))$.

Upon receiving the pain level information from agent a, agent p derives (prescribes) a medication for the patient by firing a sequence of rules from its knowledge base. The type of medication could be non-opioid, weak-opioid, or strong-opioid depending on the levels of pain. Agent p then asks agent a if the patient is responding. Depending on the response, agent p suggests either to continue the current medication or change from the current opioid to another opioid. If a patient is on the strong opioid regimen, when agent p interacts with agent r for the reassessment, agent p also asks r if there is any side effect during medication. If at any point in time agent r informs agent p about a pain crisis, then the planner will contact agent c. If agent p receives a negative (notAvailable or Busy) response from agent c then planner p contacts agent e. Agent a in the system generates information about the pain levels at different times in the interval $[1, 5]$. For example, agent a generates pain information for a patient named `Mary` who is feeling `MucositisPainOne` using the following formula:

$X^5\ \ B_a\ \ Tell(a, p, isFeeling('Mary,' MucositisPainOne))$

Similarly, the behavior of agent a is also represented, for example, using formulas of the form:
$G(B_a\ \ Ask(p, a, needAssessment('Mary)) \rightarrow X^{\leq 4} B_a\ \ Tell(a, p, isFeeling('Mary,' Mu-cositisPainSeven)))$

We verify the following properties of the system:

$$G(\ B_p\ \ Tell(a, p, isFeeling('Mary,' MucositisPainOne)) \rightarrow X^n\ \ B_p\ \ hasPaintIntensity('Mary,' BackgroundDiscomfortIn)\ )$$

the above property specifies that whenever agent a tells agent p that `Mary` is feeling `MucositisPainOne`, agent p classifies `Mary`'s pain intensity as Background Discomfort within n time steps and

$$G(\ B_p\ \ hasCarer('Mary,' John) \wedge B_p\ \ Tell(c, p, hasAcknowledgement('John,' Busy)) \wedge B_p\ \ hasPainCrisis('Mary) \rightarrow X^n\ \ B_p\ \ Tell(p, e, hasPainCrisis('Mary))\ )$$

which specifies that whenever a pain crisis occurs with a patient and agent p has received negative acknowledgment from agent c, agent p contacts the emergency agent e within n time steps.

The above properties are verified as true when the value of n is 2 in the first property, and 3 in the second property; and the model checker uses 2 seconds for each property. However, when we assign a value to n which is less than 2 in the first property, and less than 3 in the second property, the properties are verified as false and the model checker returns counterexamples. This also demonstrates the correctness of the encoding in that the model checker does not return true for arbitrary values of n. Note that in our experiment, we use discrete time steps; however, we can map the value of n into minutes, for example n equals 3 is treated as sixty minutes.

7 Related work

There has been considerable work on the Semantic Web and rule-based agents, both in AI and in the active database community. In [35], Subercaze and Maret present a semantic agent model that allows SWRL programming of agents. A Java interpreter has been developed that communicates with the Knowledge Base using the Protégé-OWL API. The prototype tool takes advantages of the Java-based domain modeling tool JADE that allows agent registration, service discovery and messages passing. The framework supports FIPA-ACL for agent communication.

In [30], Mousavi et al. present an ontology-driven reasoning system based on BDI agent model [33]. In contrast to Jadex (that utilizes an XML format to represent agents' plans, beliefs and goals), in their framework, an ontology (in an OWL format) has been used to represent agents' believes, plans and events. The Java-based tool JADE was used to implement the agents, and the Protégé OWL was used to create the ontology. To illustrate the use of the framework, a simple Mobile Workforce Brokering Systems (a multi-agent system that automates the process of allocating tasks to Mobile Workforces) was modeled for simulation.

In [34] Ruan and MacCaull present an approach to monitor healthcare workflows using a logic-based formal method. To specify the system the authors have presented FO-LTL-K a logic fusion of first order LTL and description logic. They have shown how some of the norms in [14] can be specified using FO-LTL-K, however, the paper produced no practical results. In [31] Rabbi et al. present a different approach for modeling and verifying compensable healthcare workflow with time constraints and monitors. They use NOVA workflow tool for the modeling and automated translation of the monitoring system into the specification language of DiVinE model checker to verify properties of the system. However their framework is neither ontology nor agent based.

In [13], Dekhtyar et al. studied the complexity of verification of behavior (dynamic) properties of deterministic, nondeterministic and asynchronous multi-agent systems. They considered the MAS framework based on the IMPACT architecture presented in [36]. In order to analyze complexities in MAS verification, the authors impose easy-to-formulate limitations on IMPACT agents, which lead to a polynomial time semantics of the systems. Although in this paper we do not intend to analyze the complexity of MAS verification, the work presented in [13] motivates us to explore some research in this direction.

While in the above a number of ontology-driven modeling and reasoning approaches [30, 35] have been developed for multi-agent systems, to our knowledge tools for automated formal verification for such systems are lacking.

8 Conclusions

In this paper, we proposed an approach to modeling and verifying response time guarantees of ontology-driven multi-agent rule-based systems. We use standard model checking techniques to verify interesting properties of such systems, and show how the Maude LTL model checker can be used to verify properties including response-time guarantees of the form: if the system receives a query, then a response will be produced within n time steps. We described results of experiments on a simple healthcare monitoring system. In future work, we plan to evaluate our approach on more real-life examples of Semantic Web and rule-based systems, and enhance our framework with context-aware capabilities.

Acknowledgments This work is supported by an ACEnet Post Doctoral Fellowship, an NSERC Industrial Post Graduate Fellowship, and ACOA. The computational facilities are provided by ACEnet. We would like to thank Rachel Embree and Mary Heather Jewers for the fruitful discussions about ontologies and the guidelines for the management of cancer related pain in adults [9].

References

1. Basic formal ontology. http://ontology.buffalo.edu/bfo/ (2002)
2. SNOMED-CT Systematized Nomenclature of Medicine-Clinical Terms. http://www.ihtsdo.org/snomed-ct/ (2007)

3. OWL 2 Web Ontology Language Manchester Syntax. W3C Candidate Recommendation. http://www.w3.org/TR/owl2-manchester-syntax/ (Oct 2009)
4. OWL 2 Web Ontology Language Structural Specification and Functional-style Syntax. W3C Candidate Recommendation. http://www.w3.org/TR/owl2-syntax/ (Oct 2009)
5. The Protégé ontology editor and knowledge-base framework (Version 4.1). http://protege.stanford.edu/ (July 2011)
6. Alechina, N., Logan, B., Nga, N.H., Rakib, A.: Verifying time and communication costs of rule-based reasoners. In: Proceedings of the 5th international conference on Model checking and artificial intelligence. pp. 1–14. MoChArt'08, Springer-Verlag, Berlin, Heidelberg (2008)
7. Alechina, N., Logan, B., Nga, N.H., Rakib, A.: Automated verification of resource requirements in multi-agent systems using abstraction. In: Proceedings of the 6th international conference on Model checking and artificial intelligence. pp. 69–84. MoChArt'10, Springer-Verlag, Berlin, Heidelberg (2011)
8. Bechhofer, S., van Harmelen, F., Hendler, J.A., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference, World Wide Web Consortium, recommendation rec-owl-ref-20040210 (February 2004)
9. Broadfield, L., Banerjee, S., Jewers, H., Pollett, A.J., Simpson, J.: Guidelines for the management of cancer-related pain in adults. Supportive care cancer site team, cancer care Nova Scotia, Canada. (2005)
10. Clavel, M., Eker, S., Lincoln, P., Meseguer, J.: Principles of Maude. Electronic Notes in Theoretical Computer Science. 4, 65–89 (1996)
11. Cost, R.S., Finin, T.W., Joshi, A., Peng, Y., Nicholas, C.K., Soboroff, I., Chen, H., Kagal, L., Perich, F., Zou, Y., Tolia, S.: ITtalks: A case study in the Semantic Web and DAML+OIL. IEEE Intelligent Systems 17, 40–47 (January 2002)
12. Culbert, C.: CLIPS reference manual. NASA (2007)
13. Dekhtyar, M.I., Dikovsky, A.J., Valiev, M.K.: On complexity of verification of interacting agents' behavior. Annals of pure and applied logic 141(3), 336–362 (2006)
14. Ferris, F.D., Balfour, H.M., Bowen, K., Farley, J., Hardwick, M., Lamontagne, C., Lundy, M., Syme, A., West, P.J.: A model to guide hospice palliative care: Based on national principles and norms of practice (March 2002)
15. Friedman-Hill, E.J.: Jess, the rule engine for the java platform. Sandia national laboratories (2008)
16. Glimm, B., Horridge, M., Parsia, B., Patel-Schneider, P.F.: A syntax for rules in OWL 2. In: Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009). vol. 529. CEUR (2009)
17. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proceedings of the 12th international conference on World Wide Web. pp. 48–57. ACM Press (2003)
18. Gruber, T.: A translation approach to protable ontology specifications. Knowledge Acquisition 5, 199–220 (1993)
19. Hardiker, N., Coenen, A.: A formal foundation for ICNP. Journal of Stud Health Technol Inform 122, 705–709 (2006)
20. Hendler, J.: Agents and the semantic web. IEEE Intelligent Systems 16, 30–37 (2001)
21. Horridge, M., Bechhofer, S.: The OWL API: A java API for working with OWL 2 Ontologies. In: 6th OWL Experienced and Directions Workshop (OWLED) (October 2009)

22. Horrocks, I., Patel-Schneider, P.F.: A proposal for an OWL rules language. In: Proceedings of the 13th international conference on World Wide Web. pp. 723–731. WWW '04, ACM Press (2004)
23. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web rule language combining OWL and RuleML. Acknowledged W3C submission, standards proposal research report: Version 0.6 (April 2004)
24. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. Journal of Web Semantics 3(2-3), 79–115 (2005)
25. Krötzsch, M., Maier, F., Krisnadhi, A., Hitzler, P.: A better uncle for owl: nominal schemas for integrating rules and ontologies. In: Proceedings of the 20th international conference on World wide web. pp. 645–654. ACM (2011)
26. Krötzsch, M., Mehdi, A., Rudolph, S.: Orel: Database-driven reasoning for OWL 2 profiles. In: Proceedings of the 23rd International Workshop on Description Logics. vol. 573. CEUR (2010)
27. McGuinness, D., Smith, M., Welty, C.: OWL Web Ontology Language Guide, W3C Recommendation. http://www.w3.org/TR/owl-guide/ (February 2004)
28. Motik, B., Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language: Profiles, W3C Recommendation. http://www.w3.org/TR/owl2-profiles/ (October 2009)
29. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 3, 41–60 (2005)
30. Mousavi, A., Nordin, M.J., Othman, Z.A.: An ontology driven, procedural reasoning system-like agent model,for multi-agent based mobile workforce brokering systems. Journal of Computer Science. 6, 557–565 (2010)
31. Rabbi, F., Mashiyat, A., MacCaull, W.: Model checking workflow monitors and its application to a pain management process. In: Proceedings of International Symposium on Foundations of Health Information Engineering and Systems. pp. 110–127. Johannesburg, South Africa (2011)
32. Rakib, A.: Verifying requirements for resource-bounded agents. Ph.D. thesis, The University of Nottingham. (2011)
33. Rao, A.S., Georgeff, M.P.: BDI Agents: From Theory to Practice. In: Proceedings of the First International Conference on Multi-agent Systems. pp. 312–319. The MIT Press (1995)
34. Ruan, J., MacCaull, W.: Data-aware monitoring for healthcare workflows using formal methods. In: Proceedings of The Second Workshop Knowledge Representation for Health Care(KR4HC'10). pp. 51–60. Lisbon, Portugal (2010)
35. Subercaze, J., Maret, P.: SAM - semantic agent model for swrl rule-based agents. In: Proceedings of the International Conference on Agents and Artificial Intelligence. pp. 245–248. INSTICC Press (2010)
36. Subrahmanian, V.S., Bonatti, P., Dix, J., Eiter, T., Kraus, S., Ozcan, F., Ross, R.: Heterogeneous Agent Systems. MIT Press (2000)
37. Tzafestas, S.G.: Knowledge-Based System Diagnosis, Supervision, and Control. Plenum Publishing Co. (1988)