A Formal Approach to Modelling and Verifying Resource-Bounded Context-Aware Agents

Abdur Rakib[1] and Rokan Uddin Faruqui[2]

[1] School of Computer Science
University of Nottingham, Malaysia Campus
Abdur.Rakib@nottingham.edu.my
[2] Department of Computer Science and Engineering
University of Chittagong, Bangladesh
rufaruqui@cu.ac.bd

Abstract. There has been a move of context-aware systems into safety-critical domains including healthcare, emergency scenarios, and disaster recovery. These systems are often distributed and deployed on resource-bounded devices. Therefore, developing formal techniques for modelling and designing context-aware systems, verifying requirements and ensuring functional correctness are major challenges. We present a framework for the formal representation and verification of resource-bounded context-aware systems. We give ontological representation of contexts, translate ontologies to a set of Horn clause rules, based on these rules we build multi-agent context-aware systems and encode them into Maude specification, we then verify interesting properties of such systems using the Maude LTL model checker.

Keywords: Pervasive computing, Context-awareness, Multi-agent systems, Ontology, Model checking.

1 Introduction

It is widely acknowledged that computer systems are becoming increasingly nomadic and pervasive. The vision of this next generation technology intends to provide invisible computing environments so that a user can utilize services at any time and everywhere [1]. In these systems information can be collected by using tiny resource-bounded devices, such as, e.g., PDAs, smart phones, and wireless sensor nodes. In recent years much research in pervasive computing has been focused on incorporation of context-awareness features into pervasive applications. There is an extensive body of work in adapting the Semantic Web technologies to model context-aware systems (see e.g.,[2,3,4,5]). In the pursuit of making context-aware system much more useful we need to make its various devises communicate with each other and with the surrounding environment in a cooperative manner. In achieving this goal, agent-based techniques can be seen as a promising approach for developing context-aware applications in complex domains. The concept of agents, in the setting of this paper is used to refer to autonomous reasoning agents, where agents are capable of reasoning about their behaviour (using a knowledge base and inference rules) and interactions (capable of communicating with each other). In agent-based techniques agents (devices or

P.C. Vinh et al. (Eds.): ICCASA 2012, LNICST 109, pp. 86–96, 2013.

environment) are allowed to make intelligent decisions and perform appropriate actions. E.g., communication between battery-powered sensor nodes consumes most of the available power. In order to increase the life time of sensor nodes, the amount of information broadcast to other sensor nodes should be minimised. Each sensor node (assuming modelled as an agent) should make local decisions in order to determine what information should be communicated, and to whom. E.g., instead of broadcasting all the *WindSpeed* readings, a sensor node may only send the average or the maximum or minimum of *WindSpeed* readings taken over a specified amount of time.

The main emphasis of the existing research on context-aware computing including those presented in [2,3,4,5] is how can ontologies be utilised for context-modelling, knowledge sharing and reasoning about context for pervasive computing systems. However, that is not sufficient to make context-aware systems a key feature technology that has been moving into safety-critical domains including healthcare, emergency scenarios, and disaster recovery. Moreover, in real-world context-aware agents are often resource-bounded. In this trend, to develop smarter and reliable application of context-aware systems, we need a rigorous study not only on formal representation of such systems but also their formal specification and verification. In this paper, we consider these two problems jointly. In addition to the formal representation of context-aware systems, we consider distributed problem-solving in systems of communicating context-aware 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 derive a goal. We use the Maude LTL model checker [6] to verify interesting properties of such systems.

The remainder of the paper is organized as follows. In section 2, we discuss how contexts are represented using OWL 2 RL and SWRL, and ontological context reasoning to infer higher level contexts. In section 3, we describe our model of communicating multi-agent context-aware systems. In section 4, we briefly describe specification of the multi-agent context-aware systems into Maude, and to illustrate the application of the framework in section 5 we present an example system and experimental results. We discuss related work in section 6 and conclude in section 7.

2 Ontology-Based Context Representation

In context-aware computing, the definition of context has been at the centre of different research efforts, however, a universally consented definition of context has been difficult to realise. We view context is any information that can be used to identify the status of an entity. An entity can be a person, a place, a physical or a computing object. This context is relevant to a user and application, and reflects the relationship among themselves [7].

A context can be formally defined as a *(subject,predicate,object)* triple that states a fact about the subject where — the subject is an entity in the environment, the object is a value or another entity, and the predicate is a relationship between the subject and object. E.g., we can represent a context "a disaster event has site Southern Florida" as *(DisasterEvent,hasSite,SouthernFlorida)*.

Context modelling is a well studied topic in pervasive computing and it is a process of identifying "a concrete subset of the context that is realistically attainable from sensors, applications and users and able to be exploited in the execution of the task. The context

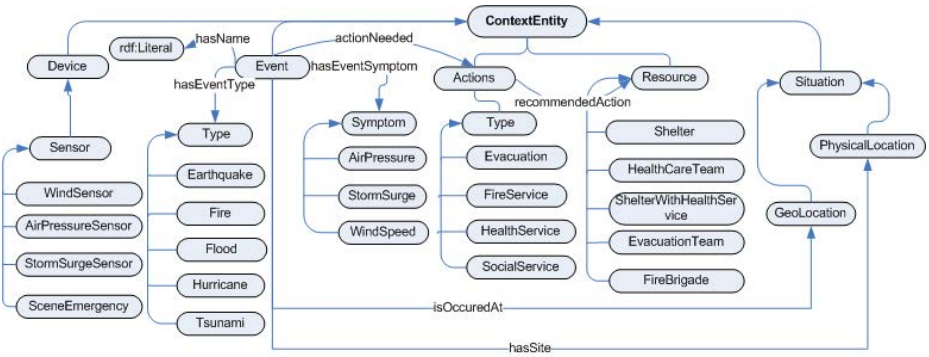

Fig. 1. A fragment of the disaster management ontology

model that is employed by a given context-aware application is usually explicitly specified by the application developer, but may evolve over time" [8]. In the literature various context modelling approaches have been proposed, however, ontology-based approach has been advocated as being the most promising one [9].

An ontology is a model of a domain that introduces vocabulary relevant to the domain and uses this vocabulary to specify the relationships among them [10]. That is, an ontology can be used to represent knowledge of a domain which gives a clear and coherent view of that domain. This facilitates the development of formal context models to share and reuse knowledge among the computational entities, such as, e.g., software agents, and provides a foundation for interoperability among the agents in a multi-agent system. Furthermore, ontology-based context modelling approach allows reasoning to infer implicit knowledge from ontologies and to generate high-level implicit contexts from the low-level explicit contexts.

Fig. 2. Individualised HurricaneEvent ontology

The common trend in ontology-based context modelling is to apply the hierarchical approach comprising of upper and domain ontologies. The upper ontology defines the high-level concepts that are common among different context-aware entities, and the domain ontology is an extension of upper ontology, defining the details of general concepts and their properties for a particular domain. For context modelling we use OWL 2 RL, a profile of the new standardization OWL 2, and based on pD^* and the description logic program (DLP) [11]. We choose OWL 2 RL because it is more expressive than the RDFS and it is suitable for the design and development of rule-based systems. To illustrate our ontology-based modelling approach for context-aware systems, we use a disaster management scenario adapted from [12]. Here we focus on the emergency response of the situation that

includes the activities designed to minimize loss of life and property. The upper ontology contains the top-level concepts for a context-aware system proposed in [13]. We use *Device*, *Event*, *Actions*, *Resource*, and *Situation* as top-level concepts to represent the context about devices (e.g., wind speed measurement sensor), events (e.g., hurricane event), rescue actions (e.g., evacuation), resources (e.g., health care team) and situation (e.g., physical location). The domain ontology can be an extension of the upper ontology for several disaster events such as earth quake, flood, tsunami, fire break, and hurricane. The domain ontology provides context about the event itself, its causes, its symptoms (e.g., wind speed, air pressure, storm surge etc.), and resources (e.g., health care team, fire brigade etc.) available to respond. Fig. 1 depicts a fragment of the disaster management ontology. An instance of the domain ontology can be, e.g., "disaster event" which is depicted in Fig 2. In this figure, we assert some low-level contexts for a disaster event namely, *hasSite(DisasterEvent, SouthernFlorida)*, *hasStormSurge(DisasterEvent,15)*, *hasWindSpeed(DisasterEvent,93)*, *hasAirPressure(DisasterEvent,93)*, and *hasName(DisasterEvent,HurricaneEvent)*. It also includes some inferred contexts derived from context-reasoning using the DL reasoner Pellet. The context reasoner classifies the disaster event as a very strong hurricane by determining high-level contexts: wind speed level, air pressure level, and storm surge level. In addition to the Pellet, we use rule-base reasoning which is discussed in the next section onwards.

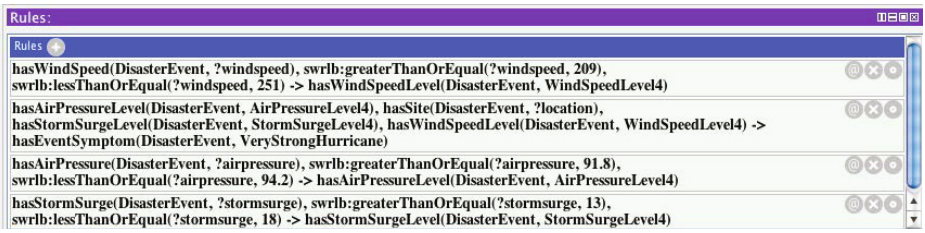

Fig. 3. Example SWRL rules

The combination of upper and domain ontologies described above, however only capture the static behaviour of a context-aware system. The context-aware systems modelled in our approach define their dynamic behaviour using Semantic Web Rule Language (SWRL). SWRL allows user to write rules using OWL concepts and its combination with OWL 2 RL provide more expressive language having greater deductive reasoning capabilities than OWL 2 RL alone. We can express more complex rule-based concepts using SWRL that cannot be modelled using OWL 2 RL, as shown in Fig. 3. Thus our approach of ontological representation of context-aware systems gives a clean ontology design based on the distinction between the static information represented using OWL 2 RL and the dynamic aspects of the systems go into the SWRL rules. We build ontologies using Protégé version 4.1 [14].

3 Multi-agent Context-Aware Systems

Pervasive computing systems which include multiple interacting devices and human users can often be usefully modelled as multi-agent systems. Non-human agents in such a system may be running a very simple program, however they are increasingly designed to exhibit flexible, adaptable and intelligent behaviour. A common methodology for implementing the latter type of agents is implementing them as rule-based reasoners. We extract Horn clause rules from ontologies to design our rule-based agents. We developed a translator that takes as input an OWL 2 RL ontology in the OWL/XML format (output file of Protégé) and translates it to a set of plain text Horn clause rules. We use the OWL API to parse the ontology and extract the set of axioms and facts. In [11] a Description Logic Mapping (DLP) mapping is given to translate an OWL 1 ontology to a set of Horn clause rules. We extended the DLP mapping to accommodate new features of OWL 2 RL. The translation of SWRL rules is straightforward because they are already in the Horn clause rule format. The translation of OWL 2 RL + SWRL to Horn clause rules is automatic and is a part of TOVRBA[15].

We adopt the model of multi-agent systems presented in [16]. A multi-agent system consists of $n_{Ag} (\geq 1)$ individual *agents* $\mathcal{A} = \{1, 2, \ldots, n_{Ag}\}$. Each agent $i \in \mathcal{A}$ has a program, consisting of Horn clause rules of the form $C_1 \wedge C_2 \wedge \ldots \wedge C_n \rightarrow C$ (derived from OWL 2 RL and SWRL), and a working memory, which contains ground atomic contexts taken from ABox representing the initial state of the system. In our model, agents share a common ontology and communication mechanism. To model communication between agents, we assume that agents have two special communication primitives $Ask(i, j, C)$ and $Tell(i, j, C)$ in their language, where i and j are agents and C is an atomic context not containing an Ask or a $Tell$. $Ask(i, j, C)$ means 'i asks j whether the context C is the case' and $Tell(i, j, C)$ means 'i tells j that context C' ($i \neq j$). 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, C) in the consequent of a rule; e.g., $C_1 \wedge C_2 \wedge \ldots \wedge C_n \rightarrow Ask(i, j, C)$ whereas agent j may have an Ask or a $Tell$ with arguments (i, j, C) in the antecedent of the rule; e.g., $Tell(i, j, C) \rightarrow C$ is a well-formed rule (we call it trust rule) for agent j that causes it to believe i when i informs it that context C is the case. No other occurrences of Ask or $Tell$ are allowed. 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.

Firing a communication rule instance with the consequent $Ask(i, j, C)$ adds the context $Ask(i, j, C)$ both to the working memory of i and of j. Intuitively, i has a record that it asked j whether context C is the case, and j has a record of being asked by i whether context C is the case. Similarly, if the consequent of a communication rule instance is of the form $Tell(i, j, C)$, then the corresponding context $Tell(i, j, C)$ is added to the working memories of both the agents i and j. The agents in the system execute synchronously. We assume that each agent executes in a separate process and that agents communicate via message passing. We also assume that there is a bound on communication for each agent i which limits agent i to at most $m_i (\geq 0)$ messages. Each agent has a communication counter, msg_i, which starts at 0 and is not allowed to

exceed the value m_i. We further assume that each agent can communicate with multiple agents in the system at the same time. At each step in the evolution of the system, each agent chooses from a set of possible actions: Rule firing a rule, Communication agents can exchange messages regarding their facts using Ask and $Tell$, and Idle which leaves its state unchanged. The actions selected by the agents are then performed in parallel and the system advances to the next state.

4 Specifying Systems in Maude

We use extended version of the TOVRBA tool [15] to translate OWL/XML rules produced by Protégé into plain text Horn clause rules of the form: $\langle n : C_1 \wedge C_2 \wedge \ldots \wedge C_n \rightarrow C \rangle$, where n is the user annotated priority of the rule. In this step the system designer identifies which agents (s)he needs to design using which rules. The designer also determines the number of agents (s)he needs to model and their possible interactions. An agent can interact with one or more agents in the system, but not necessarily every agent interacts with every other agent in the system. When the rules are classified for the agents, the multi-agent system can be implemented in Maude. The internal configuration of the rules in Maude specification has the following form: $\langle n : [t_1 : C_1] \wedge [t_2 : C_2] \wedge \ldots \wedge [t_n : C_n] \rightarrow [t : C] \rangle$, where the t_i's and t represent time stamps of contexts automatically inserted by TOVRBA. When a rule instance of the above rule is fired, the newly generated context C 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. The associated time stamp reflects when a particular context has been generated. Initially all the working memory contexts have time stamp 0 and the system moves and generates new contexts at different time points.

A multi-agent rule-based system has three components: the knowledge base (KB) which contains rules, the working memory (WM) which contains facts, and the inference engine which reasons over rules when the application is executed. In our framework, the inference engine have some reasoning strategies often used in rule-based systems [17] including rule ordering, depth, breadth, simplicity, and complexity.

In Maude specification each agent in the system has a local configuration and the (parallel) composition of all these local configurations make the global configuration of the multi-agent system. To implement the local configuration of an agent (working memory, program, agenda, reasoning strategies, message counters, time stamps etc.) we declared a number of sorts, including `Context`, `Term`, `WM`, `TimeC`, `TimeWM`, `Rule`, and `Agenda`, and define their relationships, e.g., `TimeC` is a subsort of `TimeWM`, `Rule` is a subsort of `Agenda` and so on. In addition, we use a number of Maude library modules such as `NAT`, `BOOL`, and `QID`. The local configuration of an agent i is represented as a tuple `[ A | RL | TM | M | t | msg | syn ]`, where the variables `A` and `RL` are of sort `Agenda`, `TM` is of sort `TimeWM`, `M` is of sort `WM`, and `t`, `msg`, `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 synchronisation.

In this paper we don't discuss complexity issues, however, the translation of the ontology-driven rules in Maude takes polynomial space. The rules of an agent i are defined using an operator which takes arguments as a set of contexts known as antecedents

(of sort `TimeWM`) and a single context known as consequent (of sort `TimeC`) and it returns an element of sort `Rule`. Therefore, each rule of an agent i is an element of sort `Rule`. These rules are represented using Maude equations, one equation for each rule. As an example, the rule $\langle\, 1 :\ C1 \rightarrow C2 \rangle$ can be represented as follows:

```
ceq ruleIns(A,[t1:C1] TM, M ) = <1:[t1:C1]-> [0:C2]> ruleIns(<1:
[t1:C1]-> [0:C2] A, [t1:C1] TM, M ) if (not inAgenda(<1:[t1:C1]
->[0:C2]>, A) /\ (not inWM(C2, M)) .

eq ruleIns(A, TM , M) = void-rule [owise] .
```

The inference engine is implemented using a set of Maude rules: `Generate`, `Choice`, `Apply`, `Idle`, and `Communication`. The `Generate` rule causes each agent to generate its conflict set by calling recursively rule equations like defined above. Each equation may give rise to more than one rule instance depending on the elements in working memory. To prevent the regeneration of the same rule instance, the conditional equation checks whether the rule instance and its consequent are already present in the agenda and working memory. The `Choice` rule causes each agent to apply its reasoning strategy, the `Apply` rule causes each agent to execute the rule instances selected for execution, the `Idle` rule executes only when there are no rule instances to be executed (the application of the `Idle` rule advances the cycle time of the agent i, leaving everything else unchanged), and communication among agents is achieved using the `Communication` rule. When agents communicate with each other, one agent copies the communicated fact from another agent's working memory. Copying is only allowed if the fact to be copied is not already in the working memory of the agent intending to copy.

5 Verifying Time and Communication Costs

This section shows how a context-aware multi-agent system specified in Maude can be formally verified using the Maude LTL model checker. Model checking in Maude involves a Maude specification of a system (as described in the previous section) together with a property of interest. A property is a Linear Temporal Logic (LTL) formula interpreted as a property of computations of the system (linear sequences of states generated by application of rewrite rules).

We build a multi-agent rule-based system whose rules are derived from the ontology of the disaster event scenario mentioned in Section 2. The system consists of eleven agents, Fig. 4 depicts disaster management agents and their possible interactions. The sensor agents 1, 2, and 3 are able to infer high-level contexts from sensed low-level contexts using Horn clause

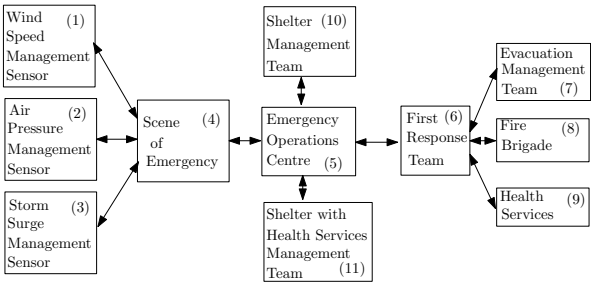

Fig. 4. Agents and their possible interactions

rules in their KB. They can classify wind, air, and storm into different levels based on wind speed, air pressure, and storm surge. E.g., agent 1's KB contains rules including the following:

⟨1 : *hasWindSpeed(DisasterEvent,?windspeed)& greaterThanOrEqual(?windspeed,209)& lessThanOrEqual(?windspeed,251) →hasWindSpeedLevel(DisasterEvent,WindSpeedLevel4)* ⟩; and

⟨2 : *hasWindSpeedLevel(DisasterEvent,WindSpeedLevel4)→ Tell(1,4,hasWindSpeedLevel(DisasterEvent,WindSpeedLevel4))* ⟩.

The first rule classifies that the disaster event has *WindSpedLevel4* if wind speed is greater than or equal to 209km/h and less than or equal to 251km/h. That is agent 1 may infer high-level context *hasWindSpeedLevel(DisasterEvent, WindSpeedLevel4)* from the low-level contexts, e.g., *hasWindSpeed(DisasterEvent, 220)*, *greaterThanOrEqual(220, 209)*, and *lessThanOrEqual(220, 251)*. In our model we classify five different levels of wind speed, air pressure, and storm surge based on NOAA/National Hurricane Center's information.[1] The second rule is a communication rule of agent 1 through which it interacts with agent 4 and sends context *hasWindSpeedLevel(DisasterEvent, WindSpeedLevel4)* when it believes that current wind speed has *WindSpeedLevel4*. Similar to the above, agent 1 has eight other rules for the other four different wind speed levels. In a similar fashion, agent 2 may infer context *hasAirPressureLevel(DisasterEvent, AirPressureLevel4)* if current air pressure is greater than or equal to 91.7kPa and less than or equal to 94.2kPa, and agent 3 may infer *hasStromSurgeLevel(DisasterEvent, StormSurgeLevel4)* if current storm surge is greater than or equal to 13ft and less than or equal to 18ft, and so on. Once these agents infer high level contexts they can interact with agent 4 and send those information. Agent 4's KB contains rules including:

⟨1 : *Tell(1,4,hasWindSpeedLevel(DisasterEvent,WindSpeedLevel4))→hasWindSpeedLevel(DisasterEvent, WindSpeedLevel4)*⟩;

⟨2 :*hasWindSpeedLevel(DisasterEvent,WindSpeedLevel4)& hasAirPressureLevel(DisasterEvent, AirPressureLevel4)& hasStormSurgeLevel(DisasterEvent,StormSurgeLevel4)& hasSite(Disaster Event,?location)→hasEventSymptom(DisasterEvent, VeryStrongHurricane)* ⟩; and

⟨3 :*hasEventSymptom(DisasterEvent,VeryStrongHurricane)& hasSite(DisasterEvent,?location) → Tell(4,5,hasOccurredAt(VeryStrongHurricane,?location))*⟩.

The first rule is a trust rule for agent 4 that causes it to believe agent 1 when agent 1 informs it that context *hasWindSpeedLevel(DisasterEvent, WindSpeedLevel4)*. Upon receiving context information from three different measurement sensor agents, using the second rule, agent 2 infers, e.g., the context *hasEventSymptom(DisasterEvent, VeryStrongHurricane)*. Using the third rule agent 4 interacts with agent 5 and informs that very strong hurricane took place at Southern Florida if it already believes context *hasSite(DisasterEvent, SouthernFlorida)* i.e., tells the context *hasOccurredAt(VeryStrongHurricane, SouthernFlorida)*. Due to space limitations, we are unable to represent example rules of other agents in the system. However, agent 5 then interacts with agent 6 and informs that an action is required against very strong hurricane that has taken place at Southern Florida. Depending on the hurricane category, agent 6 interacts with various services including agents 7, 8, and 9. Agent 7 then reply back to

[1] `http://www.nhc.noaa.gov/aboutsshws.php`
`http://hypertextbook.com/facts/StephanieStern.shtml`

agent 6 that immediately n number simple shelters and m number shelter with health services are needed. Agent 6 relays these information to agent 5. Agent 5 then enquires to agents 10 and 11 whether required number shelters are available, and receives shelter availability replies. Similarly, depending on the hurricane category agent 8 and agent 9 might take different actions including Land Fire Brigade, Air Tanker, Boat Ambulance and Air Ambulance Services. In order to model this scenario we have used 122 Horn clause rules distributed to the agents. E.g., the KB of each of the measurement sensor agents contains 10 rules, agent 4 is modelled using 25 rules, agent 5 is modelled using 16 rules, and so on. We verified a number of interesting resource-bounded properties of the system including the following:

$$G(B_4\, hasSite(DisasterEvent, SouthernFlorida)) \\ \wedge B_4\, Tell(1, 4, hasWindSpeedLevel(DisasterEvent, WindSpeedLevel4)) \\ \wedge B_4\, Tell(2, 4, hasAirPressureLevel(DisasterEvent, airPressureLevel4)) \\ \wedge B_4\, Tell(3, 4, hasStormSurgeLevel(DisasterEvent, stormSurgeLevel4)) \\ \rightarrow X^n \;\; B_4\, Tell(4, 5, hasOccurredAt(VeryStrongHurricane, SouthernFlorida)))$$

the above property specifies that whenever agents 1, 2, and 3 tell agent 4 that the wind speed, air pressure, and storm surge have reached fourth level at the disaster event site Southern Florida, within n time steps agent 4 informs agent 5 that very strong hurricane has occurred at Southern Florida (where B_i for each agent i is a syntactic doxastic operator used to specify agent i's 'beliefs' or the contents of its working memory, and X^n is the concatenation of n LTL next operators X) and

$$G(B_5\, Tell(4, 5, hasOccurredAt(VeryStrongHurricane, SouthernFlorida)) \\ \rightarrow X^n \;\; B_5\, Ask(5, 10, hasAvailableShelter(SouthernFlorida, 10000)) \wedge msg_5{}^{=m})$$

which specifies that whenever agent 5 receives information from agent 4 that very strong hurricane has occurred at Southern Florida, within n time steps agent 5 asks agent 10 if shelter is available for 10000 homeless people at Southern Florida while exchanging m messages ($msg_5{}^{=m}$ states that the value of agent 5's communication counter is m).

The above properties are verified as true when the value of n is 5 in the first property, and the values of n and m are 18 and 5 in the second property; and the model checker uses 1 seconds for each property. However, the properties are verified as false and the model checker returns counterexamples when we assign a value to n which is less than 5 in the first property, and values to n and m which are less than 18 and 5 in the second property. This also demonstrates the correctness of the encoding in that the model checker does not return true for arbitrary values of n and m.

6 Related Work

We present a brief existing research on context-aware systems concentrating on the ontology-based approaches which have influenced the work presented in this paper. In [2], Wang et al. proposed OWL encoded context ontology (CONON) for modelling and reasoning in pervasive systems. In addition to the ontological reasoning they use a set of user-defined FOL rules to reason over data, i.e., LP reasoning combined with DL reasoning in separate tasks. In contrast, OWL 2 RL and SWRL have been used in our work which give an expressive ontology language to capture the knowledge of complex context-aware systems. In [4], Keßler et al. has shown the usefulness of SWRL

rules in modelling context-aware scenarios, and how those rules can be used for personalised mappings between the numeric sensor world and information stored in ontologies. The main focus of their work is context-aware instantiation based on SWRL rules and built-ins. In [5], OWL ontologies are used to model context-aware systems, the authors exploited classes and properties from ontologies to write rules in Jess to derive multi-agent rules based system. Thus their modelling part of the system only reflects the static behaviour. In contrast, our ontology-based modelling captures both static and dynamic behaviour of the system using OWL 2 RL and SWRL. A prototype of context management model is presented in [3] that supports collaborative reasoning in a multi-domain pervasive context-aware application. The model facilitates the context reasoning by providing structure for contexts, rules and their semantics. However, none of the existing approaches discussed above considers formal specification and verification of context-aware systems.

7 Conclusions and Future Work

In this paper, we proposed a formal approach to modelling and verifying context-aware systems. We gave ontological representation of contexts and shown how we build context-aware systems as multi-agent systems, specify them in Maude and ultimately verify their interesting properties using model checking. Our approach gives a clean ontology design based on the distinction between the static information represented using OWL 2 RL and the dynamic aspects of the systems go into the SWRL rules. In future work, we would like to present a formal logical model for context-aware systems based on temporal epistemic description logics. In addition to the time and communication resources, we will also consider space requirement for reasoning.

References

1. Weiser, M.: The computer for the 21st century. ACM SIGMOBILE Mobile Computing and Communications Review - Special Issue Dedicated to Mark Weiser 3(3), 3–11 (1999)
2. Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K.: Ontology based context modeling and reasoning using owl. In: PerCom Workshops 2004, pp. 18–22 (2004)
3. Ejigu, D., Scuturici, M., Brunie, L.: An ontology-based approach to context modeling and reasoning in pervasive computing. In: PerCom Workshops 2007, pp. 14–19 (2007)
4. Keßler, C., Raubal, M., Wosniok, C.: Semantic Rules for Context-Aware Geographical Information Retrieval. In: Barnaghi, P., Moessner, K., Presser, M., Meissner, S. (eds.) EuroSSC 2009. LNCS, vol. 5741, pp. 77–92. Springer, Heidelberg (2009)
5. Esposito, A., Tarricone, L., Zappatore, M., Catarinucci, L., Colella, R., DiBari, A.: A Framework for Context-Aware Home-Health Monitoring. In: Sandnes, F.E., Zhang, Y., Rong, C., Yang, L.T., Ma, J. (eds.) UIC 2008. LNCS, vol. 5061, pp. 119–130. Springer, Heidelberg (2008)
6. Eker, S., Meseguer, J., Sridharanarayanan, A.: The Maude LTL Model Checker and Its Implementation. In: Ball, T., Rajamani, S.K. (eds.) SPIN 2003. LNCS, vol. 2648, pp. 230–234. Springer, Heidelberg (2003)
7. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. 5(1), 4–7 (2001)
8. Henricksen, K.: A Framework for Context-Aware Pervasive Computing Applications. PhD thesis, The University of Queensland (2003)

9. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing 2(4), 263–277 (2007)
10. Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition - Special issue: Current issues in knowledge modeling 5(2), 199–220 (1993)
11. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: WWW 2003, pp. 48–57. ACM Press (2003)
12. Rinner, C.: Multi-criteria evaluation in support of emergency response decision-making. In: Joint CIG/ISPRS Conference on Geomatics for Disaster and Risk Management (2007)
13. Baumgartner, N., Retschitzegger, W.: A survey of upper ontologies for situation awareness. In: Proceedings of the 4th IASTED International Conference on Knowledge Sharing and Collaborative Engineering, pp. 1–9 (2006)
14. Protégé: The Protégé ontology editor and knowledge-base framework (Version 4.1) (July 2011), `http://protege.stanford.edu/`
15. Rakib, A., Faruqui, R.U., MacCaull, W.: Verifying Resource Requirements for Ontology-Driven Rule-Based Agents. In: Lukasiewicz, T., Sali, A. (eds.) FoIKS 2012. LNCS, vol. 7153, pp. 312–331. Springer, Heidelberg (2012)
16. Alechina, N., Logan, B., Nga, N.H., Rakib, A.: Verifying Time and Communication Costs of Rule-Based Reasoners. In: Peled, D.A., Wooldridge, M.J. (eds.) MoChArt 2008. LNCS, vol. 5348, pp. 1–14. Springer, Heidelberg (2009)
17. Culbert, C.: CLIPS reference manual. NASA (2007)