Modeling and verifying context-aware non-monotonic reasoning agents

Abdur Rakib
School of Computer Science
University of Nottingham
Malaysia Campus
Email: Abdur.Rakib@nottingham.edu.my

Hafiz Mahfooz Ul Haque
School of Computer Science
University of Nottingham
Malaysia Campus
Email: khyx2hma@nottingham.edu.my

***Abstract*—This paper complements our previous work on formal modeling of resource-bounded context-aware systems, which handle inconsistent context information using defeasible reasoning, by focusing on automated analysis and verification. A case study demonstrates how model checking techniques can be used to formally analyze quantitative and qualitative properties of a context-aware system based on message passing among agents. The behavior (semantics) of the system is modeled by a term rewriting system and the desired properties are expressed as LTL formulas. The Maude LTL model checker is used to perform automated analysis of the system and verify non-conflicting context information guarantees it provides.**

***Keywords*—Formal modeling, Model checking, Context-aware systems, Rule-based reasoning, Defeasible reasoning, Multi-agent systems.**

I. Introduction

Context-aware computing systems have been emerging as anticipated solutions to many social complex problems, including e.g., health care [1], [2], [3]. A system is said to be context-aware if it can extract, interpret, and is able to adapt its behavior to the current context of use [4]. These systems which include multiple interacting devices and human users can often be usefully modeled as multi-agent reasoning 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 behavior. A common methodology for implementing the latter type of agents is implementing them as rule-based reasoning agents. However, while the incorporation of reasoning abilities into agents brings great benefits in terms of flexibility and ease of development, these approaches also raise new challenges for the system developer, e.g., to perform system analysis and ensure the correctness of system designs. These problems become even more challenging when the system being designed or analyzed consists of several communicating agents which exchange information via messages. In practice, system failures may happen with various reasons including e.g., a false alarm could be caused by a wrong reasoning, a system produces conflicting (inconsistent) contextual information, and/or a system's reasoning process takes undesired time to produce the desired contexts, among others. There are many cases where the time taken to do the reasoning is of critical importance. As an example, in a multi-agent context-aware system, an agent may be able to derive reasonably correct contexts (e.g., a health planner agent can infer a patient's current status based on the contextual information it has received from other heart rate and/or blood pressure measurement sensor agents) and sends the contextual information to another agent (e.g., patient's caregiver) to reach its goal, but if the overall reasoning and interaction take too long the result may be irrelevant, e.g., patient might already be in a very dangerous condition or even die before any action can be taken. These defects could only be exhibited during system deployment. This is because it is often impossible to capture all execution scenarios at development state. Thus, there is a need of a systematic formal approach to their specification and verification which can allow addressing these problems. Namely, how to ensure the correctness of context-aware rule-based designs (will a system produce the correct output for all legal inputs), termination (will a system produce an output at all) and response time (how much computation a system have to do before it generates an output). In this paper, we complement our previous work on resource-bounded non-monotonic context-aware systems [5], which handle inconsistent context information using defeasible reasoning, by focusing on automated analysis and formal verification using model checking techniques. More specifically, we show how a $\mathcal{L}_{\mathcal{DROCS}}$ [5] model can be encoded using a standard model checker such as for example the Maude LTL model checker [6] and its interesting properties can be verified.

The rest of the paper is structured as follows. In Section II, we briefly discuss context and a context-modeling approach. In Section III, we describe a formal context-aware system modeling framework. In Section IV, we briefly review Maude rewriting system and LTL model checking and show how we encode a $\mathcal{L}_{\mathcal{DROCS}}$ model in Maude. In Section V, we model an illustrative example system and verify its interesting properties, in Section VI we present related work, and conclude in Section VII.

II. Context and a context-modeling approach

In context-aware computing, even though a good deal of research has been carried out on this topic, the term *context* is not yet a well defined concept and more often researchers define this term in various ways to fit into their projects. Schilit et al. [7] have used the term relatively long ago, who define context as specific entities, such as location or object. In [8], Brown et al. define context as location, identity of nearby people, time of a day, among others. Other researchers looked

at context from a more conceptual point of view, and not only define the context to characterize the status of an entity but also emphasize on the relationships and structure of contextual information. For example, in [9] Dey et al. define context as information that characterize the situation of an entity. In this paper, we also understand under the term *context* 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.

A context can be formally defined as a subject, predicate, and object triple $\langle subject, predicate, object\rangle$ 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. According to [9], *"if a piece of information can be used to characterize the situation of a participant in an interaction, then that information is context"*. For example, we can characterize user's current status of a context-aware system based on the contexts *"Mary has fever categorized as High"* as $\langle Mary, hasFever, High\rangle$ and *"Mary has a carer named Fiona"* as $\langle Mary, hasCarer, Fiona\rangle$. Here, the caregiver Fiona of a patient Mary is dynamically identified based on the care status of Fiona. These contexts can be written using first order formulas as *hasFever('Mary, 'High)* and *hasCarer('Mary, 'Fiona)* respectively. Here and for the rest of this paper constants are preceded by a single quote, and ' and ′ have the same meaning.

In the literature, various techniques have been proposed to develop context-aware systems, including rule-based techniques [10], [3], [11]. In rule-based techniques, a context-aware system composed of a set of rule-based agents, and firing of rules that infer new contexts determine context changes and represent overall behavior of the system. In this work, we model context-aware systems as multi-agent rule-based defeasible reasoning systems. In order to model contexts and rules we use ontological approach. An ontology can represent a model of a domain of discourse that introduces a vocabulary to specify the concepts relevant to the domain and their relationships. The logic behind ontological knowledge representation is known as description logic (DL). The ability to model a domain and the decidable computational characteristics make DLs the basis for the widely accepted ontology languages such as OWL [12]. For context modeling we use OWL 2 RL, a profile of the new standardization OWL 2, and based on pD^* [13] and the description logic program (DLP) [14]. We choose OWL 2 RL because it is more expressive than the RDFS and suitable for the design and development of rule-based systems. An OWL 2 RL ontology can be translated into a set of Horn clause rules based on DLP technique [14]. Furthermore, we express more complex rule-based concepts using SWRL [15] which allow us to write rules using OWL concepts.

III. A FORMAL CONTEXT-AWARE SYSTEM MODELING FRAMEWORK

In our framework, we consider systems having constraint on various resources namely time, memory, and communication. This is because many context-aware systems often run on tiny resource-bounded devices including PDAs, mobile phones, smart phones, GPS system, and wireless sensor nodes. These devices usually operate under strict resource constraints, e.g., battery energy level, memory, processor, and quality of wireless connection. In our earlier work [5], we have presented a formal framework for modelling context-aware systems and developed a logic $\mathcal{L}_{DROCS}$ which extends the temporal logic CTL^* with belief and communication modalities and incorporates defeasible reasoning [16] technique. The logic $\mathcal{L}_{DROCS}$ allows us to describe a set of context-aware non-monotonic rule-based reasoning agents with bounds on memory and communication resources. We provided an axiomatization of the logic and proved it is sound and complete, and using a simple example we have shown how we can express some interesting resource-bounded properties of a desired system [5]. Each agent's memory usage is modeled as the maximal number of contexts to be stored in the agent's memory at any given time. That is, we assume that each agent in a system has bounded memory size which allows maximal number of contexts to be stored at any given time. Similarly, each agent has a communication counter, which starts with value 0 and incremented by 1 each time while interacting (sending/receiving a message) with other agents, and is not allowed to exceed a preassigned threshold value.

We briefly describe the systems which can be modeled using $\mathcal{L}_{DROCS}$. We model a context-aware system as a multi-agent defeasible reasoning system which consists of $n_{Ag}(\geq 1)$ individual agents $A_g = \{1, 2, ..., n_{Ag}\}$. Defeasible reasoning is a simple rule-based reasoning technique that has been used to reason with incomplete and inconsistent information [17]. A defeasible logic theory consists of a collection of rules that reason over a set of facts to reach a set of defeasible conclusions. It also supports priorities among rules to resolve conflicts. Each agent $i \in A_g$ has a program, consisting of a finite set of strict and defeasible rules, and a working memory, which contains facts (current contexts). If an agent i has a rule:

Patient(?p), hasFever(?p, 'High) $\rightarrow$ *hasSituation(?p, 'Emergency)*

and the contexts *Patient('Mary), hasFever('Mary, 'High)* are in the agent's working memory and *hasSituation('Mary, 'Emergency)* is not in the agent's working memory in state s, then the agent can fire the rule which adds the context *hasSituation('Mary, 'Emergency)* to the agent's working memory in the successor state s'. While deriving this new context, an existing context in the agent's working memory may get overwritten, and this happens if agent i's memory is full or a contradictory context arrives in the memory (even if the memory is not full). We say that two contexts are contradictory iff they are complementary with respect to $\sim$, for example, *hasSituation('Mary, 'Emergency)* and $\sim$*hasSituation('Mary,'Emergency)* are contradictory contexts. Whenever newly derived context arrives in the memory, it is compared with the existing contexts to see if any conflict arises. If so then the corresponding contradictory context will be replaced with the newly derived context, otherwise an arbitrary context will be removed if the memory is full. For example, in this case *hasSituation('Mary, 'Emergency)* will be a contradictory context if $\sim$ *hasSituation('Mary, 'Emergency)* is present in the agent's working memory, a more detailed explanation can be found in [5]. Note, however, that later in this paper in the Maude encoding we use *Not(hasSituation('Mary,*

'Emergency)) instead of $\sim$ *hasSituation('Mary, 'Emergency)* for technical reasons.

In addition to firing rules, agents can exchange messages regarding their current contexts. To model communication between agents, we assume that agents have two special communication primitives $Ask(i, j, P)$ and $Tell(i, j, P)$ in their language, where i and j are agents and P is an atomic context not containing an Ask or a $Tell$. $Ask(i, j, P)$ means 'i asks j whether the context P is the case' and $Tell(i, j, P)$ means 'i tells j that context P' ($i \neq j$).

The exchange of information between agents work like this: if an $Ask(i, j, P)$ (or a $Tell(i, j, P)$) is in agent i's working memory in state s, $Ask(i, j, P)$ (or $Tell(i, j, P)$) is not in the working memory of agent j, and agent j has not exceeded its communication bound then in the successor state s', $Ask(i, j, P)$ (or $Tell(i, j, P)$) can be added to agent j's working memory and its communication counter incremented. This action may also overwrite agent j's memory as discussed above. We view the process of producing new contexts from existing contexts as a sequence of states of an agent, starting from an initial state, and producing the next state by one of the following actions:

- Rule — firing a matching rule instance in the current state (possibly overwriting a context from the previous state);
- Copy — if agent i has an $Ask(i, j, P)$ (or a $Tell(i, j, P)$) in its current state, then agent j can copy it to its next state (possibly overwriting a context from the previous state);
- Idle — which leaves its configuration unchanged.

That is, each transition (result of an action) corresponds to a single execution step and takes an agent from one state to another. States consist of the rules, facts (contexts), and other resource counters of the agent. A *step* of the whole system is composed of the actions of each agent, in parallel. We measure time requirements for a problem as the number of such system steps. The key idea underlying the logical approach $\mathcal{L}_{DROCS}$ of context-aware systems is to define a formal logic that axiomatizes the set of transition systems, and it is then used to state various *qualitative* and *quantitative* properties of the systems. For example, a *qualitative* property could be *"Can an agent have inconsistent beliefs (contradictory contexts in its working memory)"*, and *quantitative* properties could be *"an agent will always derive context φ in t time steps while exchanging fewer than n messages"* or *"every request of an agent i will be responded by agent j in t time steps"*, among others.

IV. Maude rewriting system and formal verification

In this section, we present the basic foundation of Maude following [18] and give an overview of Maude LTL model checking. In Maude, a rewriting theory $\mathcal{R}$ = (Σ, E, R), consists of a signature Σ, a set E of equations, and a set R of rules. The static part of a system is specified in an equational sub-logic of rewriting logic (membership equational logic) by means of equations E. The system dynamics (concurrent transitions or inferences) is specified by means of rules R that rewrite terms, representing parts of the system, into other terms. The rules in R are applied *modulo* the equations in E. Maude computes normal form of a term by applying equations from left to right iteratively, then an applicable rewrite rule is arbitrarily chosen and applied from left to right. Thus, data types are defined algebraically by equations and the dynamic behavior of a system is defined by rewrite rules which describe how a part of the state can change in one step. A rewrite theory is often non-deterministic and could exhibit many different behaviors.

In Maude, a term is either a constant, a variable, or the application of an operator to a list of argument terms. A ground term is a term containing no variables, but only constants and operators. Unconditional equations are declared using the keyword `eq`, followed by an (optional) `[<LabelName>] :`, followed by a `term` (its left hand side), the equality sign `=`, then a `term` (its right hand side), optionally followed by a list of statement attributes.

```
eq [<LabelName>]:<Term-1>=<Term-2>[<OptionalStatement
Attributes>] .
```

The general form of conditional equations is the following:

```
ceq [<LabelName>]:<Term-1> = <Term-2> if <EqCond-1>
/\ ... /\<EqCond-k> [<OptionalStatementAttributes>] .
```

In Maude equations, variables appearing in the right-hand side term must also appear in its left-hand side term. Unconditional rules are declared using the keyword `rl`, followed by an (optional) `[<LabelName>] :`, a `term` (its left hand side), the Rightarrow sign `=>`, then a `term` (its right hand side).

```
rl [<LabelName>] : <Term-1> => <Term-2> .
```

Conditional rules are declared using the following syntax:

```
crl [<LabelName>]:<Term-1>=><Term-2> if <RuleCond-1>
/\.../\<RuleCond-k> .
```

The fundamental concept of Maude is the *module*, which represents the basic units of specification and programming. A module is essentially a collection of sorts and a set of operations on these sorts. There are two kinds of modules: functional modules and system modules. Each module is declared with the key terms:

```
fmod <ModuleName> is
  <Declarations And
    Statements>
edfm
```

```
mod <ModuleName> is
  <Declarations And
    Statements>
endm
```

where a functional module begins with `fmod` keyword and ends with `endfm` keyword, and a system module begins with `mod` keyword and ends with the keyword `endm`. The `<ModuleName>` represents the name of the module, and the body of a module `<DeclarationsAndStatements>` represents all the declarations and statements in between the beginning of the module and the end of it. The body of a functional module `<DeclarationsAndStatements>` defines data types and operations on them by means of equational theory E only. In contrast, the body of a system module `<DeclarationsAndStatements>` specifies a rewrite theory, which contains an equational theory E plus rewriting rules R. Like any other model checking tool, verification in Maude requires a system specification and a property specification. The system specification is provided by

a rewrite theory, whereas the property specification is given by LTL formulas.

We chose the Maude LTL model checker because it can model check systems whose states involve arbitrary algebraic data types. The only assumption is that the set of states reachable from a given initial state is finite. This simplifies modeling of the agents (first-order) rules and reasoning strategies. For example, the variables appear in a rule can be represented directly in the Maude encoding, without having to generate all ground instances resulting from possible variable substitutions.

A. Maude encoding

We take advantage of Maude's modular structuring mechanisms to implement our context-aware system (a $\mathcal{L}_{DROCS}$ model). We construct a generic functional module and a set of functional and system modules to represent the system. Each agent in the system has a configuration (local state) and the composition of all these configurations (local states) make the configuration (global state) of the system. The types necessary to implement the local state of an agent (working memory, rule-based program, reasoning strategy, message counters, memory bound, timestep etc.) are declared in a generic agent configuration functional module and its structure is given in Listing 1.

```
fmod AgentConfigModule is
  protecting NAT .
  protecting BOOL .
  protecting QID .
  sorts Constant Context Term Rule Agenda WM TimeC .
  sorts TimeWM Config .
  subsort Context < WM .
  subsort Rule < Agenda .
  subsort Qid < Constant .
  subsort TimeC < TimeWM .
  subsorts Constant < Term .
  ops void rule : -> Context .
  op [_ : _] : Nat Context -> TimeC .
  op _ _ : WM WM -> WM [comm assoc] .
  op _ _ : TimeWM TimeWM -> TimeWM [comm assoc] .
  op _ _ : Agenda Agenda -> Agenda [comm assoc] .
  op <_ : _->_> : Nat TimeWM TimeC -> Rule .
  op Ask : Nat Nat Context -> Context .
  op Tell : Nat Nat Context -> Context .
  op Not : Context -> Context .
  var c : Context .
  var M : WM .
  var C : TimeC .
  var nz : NzNat .
  var TM : TimeWM .
  ---Checking if a context is in the working memory-
  op inWM : Context WM -> Bool .
  eq inWM(c, c ) = true .
  eq inWM(c, c M) = true .
  eq inWM(c, M) = false [owise] .

  ---End Checking if a context is in the working
  ---memory-
  ------------------------------------------------
    .
    .
    .
endfm
```

Listing 1. Sorts declaration and their relationships

A number of Maude library modules such as `NAT`, `BOOL`, and `QID` have been imported into the `AgentConfigModule` 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`. A context is declared as an operator whose arguments are of sort `Term`, and returns an element of sort `Context`. Therefore, the arguments of a Context may contain constants and variables all of which are of sort `Term`. The sort `Context` 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:

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

This operation is in mixfix notation and it is commutative and associative. This means that working memory elements are a set of Contexts whose order does not matter. In order to maintain time stamp for each Context, a sort `TimeC` is declared whose elements are of the form `[ t : C ]`, where `t` represents the time stamp of context `C` that indicating when that Context was added to the working memory. The sort `TimeC` 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 commutative and associative:

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

Note that updating of `WM` and `TimeWM` take place simultaneously, for example, whenever a context `C` is added to `WM` the corresponding element `[ t : C ]` is also added to `TimeWM` for an appropriate time cycle `t`. Context time stamps are maintained to implement reasoning strategies. In $\mathcal{L}_{DROCS}$, we have used only rule priority strategy to resolve the conflicting rule instances, however, in this encoding we have also implemented other strategies often used in rule-based systems such as Depth strategy, Breadth strategy, Specificity strategy (simplicity), and Specificity strategy (complexity). Different agents in the system may use different types of reasoning strategy.

The rules of each agent are defined using an operator which takes as arguments a sort `Nat` specifying the priority, a set of contexts (of sort `TimeWM`) specifying the antecedents of the rule and a single context (of sort `TimeP`) specifying the consequent, and returns an element of sort `Rule`. 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 commutative and associative:

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

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 rules *<10 : P1(?x), P2(?x, ?y) →P3(?y)>*, and *<11:P3(?x) → TELL(1,2, P3(?y))>* of agent 1 (say) can be represented as follows (the angle brackets are used here to indicate the beginning and ending of a rule):

```
ceq ruleIns1(A,[t1:P1(?x)] [t2:P2(?x,?y)] TM, M ) =
<10 : [t1: P1(?x)] [t2:P2(?x,?y)]->[0:P3(?y)]>
ruleIns1(<10:[t1: P1(?x)] [t2:P2(?x,?y)]->[0:P3(?y)]>
A,[t1:P1(?x)] [t2:P2(?x,?y)] TM,M) if (not inAgenda(
<10 : [t1: P1(?x)] [t2:P2(?x,?y)]->[0:P3(?y)]>, A)/\
(not inWM(P3(?y), M)) .


ceq ruleIns1(A,[t1:P3(?x) ] TM, M ) =
<11 : [t1:P3(?x) ] -> [0: Tell(1,2, P3(?x))]>
ruleIns1(<11 :[t1:P3(?x) ]->[0:Tell(1,2,P3(?x))]>
A, [t1:P3(?x)] TM, M ) if (not inAgenda(
<11 :[t1:P3(?x) ]->[0: Tell(1,2, P3(?x))]>, A)/\
(not inWM( Tell(1,2,P3(?x)), M)) .

eq ruleIns1(A, TM , M) = void-rule [owise] .
```

In the rule, the numbers 10 and 11 represent rule priorities and the place holders $t1$ and $t2$ represent time stamp of

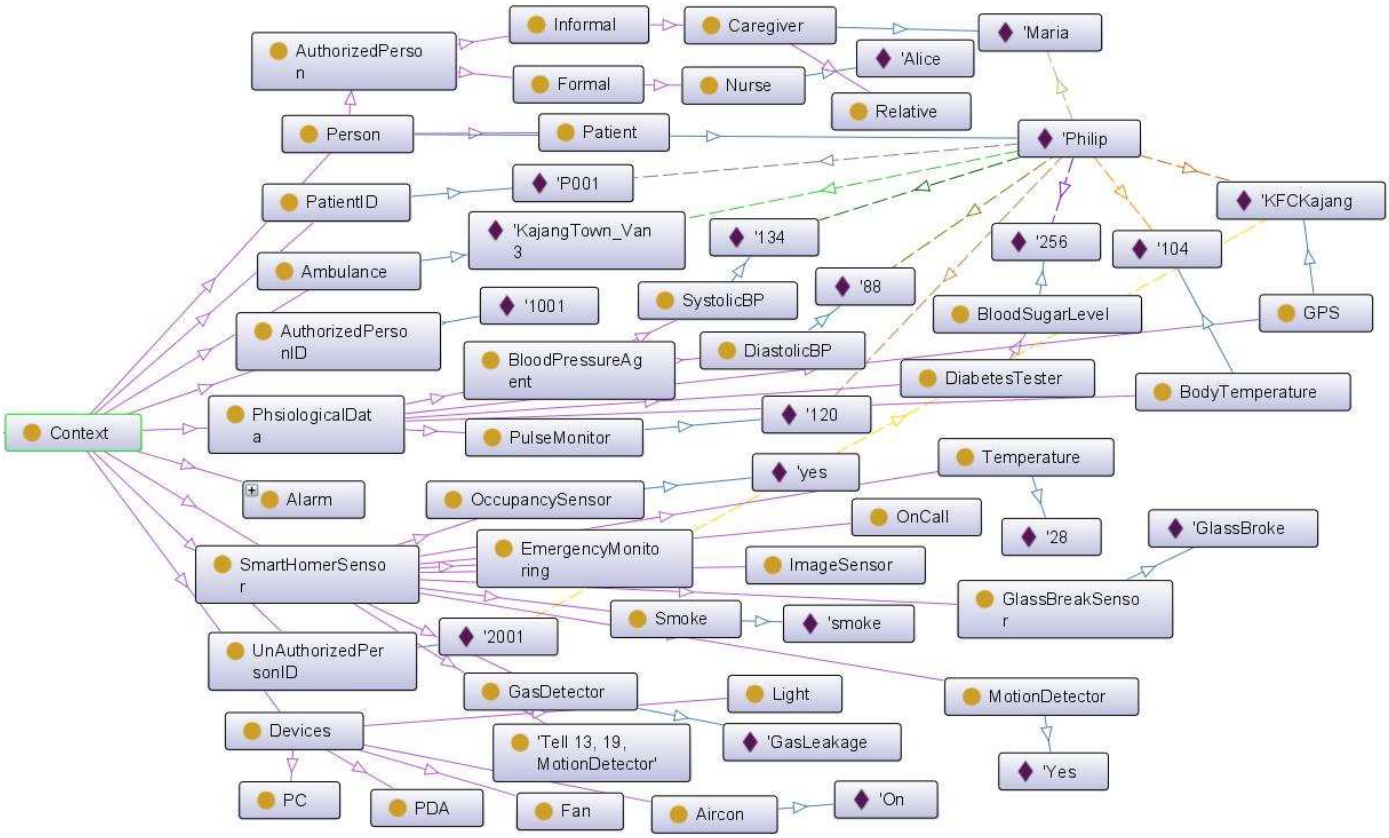

Fig. 1. A partial view of the context modelling ontology

Rules:

BodyTemperature(?temp), Person(?p), hasBodyTemperature(?p, ?temp), greaterThan(?temp, "95"), lessThan(?temp, "99") -> hasFever(?p, "Normal")

hasDBCategory(?p, "Hyperglycaemia") -> 'Tell 3, 1, hasDBCategory'(?p, "Hyperglycaemia")

Temperature(?temp), greaterThan(?temp, "18"), lessThan(?temp, "25") -> hasTemperature(?temp, "Normal")

OccupancySensor(?x) -> SmartHomerSensor(?x)

AuthorizedPersonID(?apid), Person(?p), hasAuthorizedPersonID(?p, ?apid) -> isAuthorizedPerson(?p, "Yes")

'Tell 7, 10, hasWarningSign'(?p, ?loc) -> hasWarningSign(?p, ?loc)

'Tell 8, 17, BurglarAlarm'(?x) -> Alarm(?x)

'Tell 2, 1, hasBloodPressure'(?p, "Normal") -> hasBloodPressure(?p, "Normal")

'Tell 10, 6, hasAmbulanceCallFor'(?p, ?loc) -> hasAmbulanceCallFor(?p, ?loc)

hasBloodPressure(?p, "Prehypertension") -> 'Tell 2, 1, hasBloodPressure'(?p, "Prehypertension")

'Tell 9, 8, hasGPSLocation'(?p, ?loc) -> hasGPSLocation(?p, ?loc)

Caregiver(?c), hasAlarmFor(?p, ?loc), hasCaregiver(?p, ?c) -> logAlarm(?c, ?p)

'Tell 2, 1, hasBloodPressure'(?p, "Hypotension") -> hasBloodPressure(?p, "Hypotension")

hasOccupancy(?p, "No") -> 'Tell 19, 18, hasOccupancy'(?p, "No")

'Tell 3, 1, hasDBCategory'(?p, "Hypoglycaemia") -> hasDBCategory(?p, "Hypoglycaemia")

BodyTemperature(?temp), Person(?p), hasBodyTemperature(?p, ?temp), greaterThanOrEqual(?temp, "99"), lessThan(?temp, "101") -> hasFever(?p, "High")

DiastolicBP(?dbp), Person(?p), SystolicBP(?sbp), hasDiastolicBP(?p, ?dbp), hasSystolicBP(?p, ?sbp), greaterThan(?dbp, "90"), greaterThan(?sbp, "140"), lessThan(?dbp, "100"), lessThan(?sbp, "160") -> hasBloodPressure(?p, "Stage1hypertension")

Fig. 2. Some rules of the smart environment ontology

the corresponding context. Each equation may give rise to more than one rule instance depending on the current contexts in the agent's 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 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 those defined above.

```
rl[Generate]:<S1[A1|RL1|TM1|M1|t1|m1|msg1|1]1S||C,com>
=>
< S1[ ruleIns1(A1, RL1,TM1,M1) A1| RL1|TM1|M1|t1|m1|
msg1|1]1S || C, com> .
```

In the `Generate` rule above, `S1[ A1 | RL1 | TM1 | M1 | t1 | m1 | msg1 | 1 ]1S` represents local state of agent 1 and the variable `C` of type `Config` represents local states of all other agents in the system, and all these states are composed using the || operator. The structures of the individual and multi-agent modules are shown in Listing 2.

```
fmod Agent-i is
  protecting AgentConfigModule .
  op Si[_|_|_|_|_|_|_|_]iS : Agenda Agenda TimeWM
                        WM Nat Nat Nat Nat -> Config .
     .
     .
     .
endfm

 mod MultiAgentSystem is
  protecting Agent-i .
  .
  .
  .
  sort masConfig .
  sort Phase .
  ops com exec : -> Phase .
  var phase : Phase .
  op _||_ : Config Config -> Config [comm assoc] .
  op <_,_> : Config Phase -> masConfig [ctor] .
  .
  .
  .
endm
```

Listing 2. Structure of multi-agent module

Similarly, 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 an 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 context from another agent's working memory. Copying is only allowed if the context to be copied is not already in the working memory of the agent intending to copy and the agent has not exceeded its communication bound.

V. A SMART ENVIRONMENT EXAMPLE SYSTEM

We develop a multi-agent non-monotonic context-aware system whose rules are derived from a smart environment domain ontology. The example scenario is adopted from [19], [20], [21], which is further extended based on the system users' requirements. This example system aims to facilitate residents in an intelligent home care environment that address residents' needs based on the current contexts. The aim is to create an automated assisted living environment for needy people to live a safe life and provide ease, comfort and security to them. In this system design we consider a number of intelligent context-aware agents to monitor the current status of a person and the home environment. For example, a number of essential health care devices are considered to monitor a patient's vital information, which update status based on the current contexts. In case of a critical condition, for example, a patient has very high fever or patient's pulse rate is abnormal then an emergency alarm may be activated to alert caregivers to take appropriate actions. This smart home environment also considers some security agents to monitor unauthorized persons or prohibited activities at home. Fig. 1 and Fig. 2 depict partial view and some rules of the smart environment ontology, Fig. 3 depicts an individualized smart environment ontology, and Fig. 4 depicts smart space context-aware agents and their possible interactions. However, the complete ontology and rules can be found online[1].

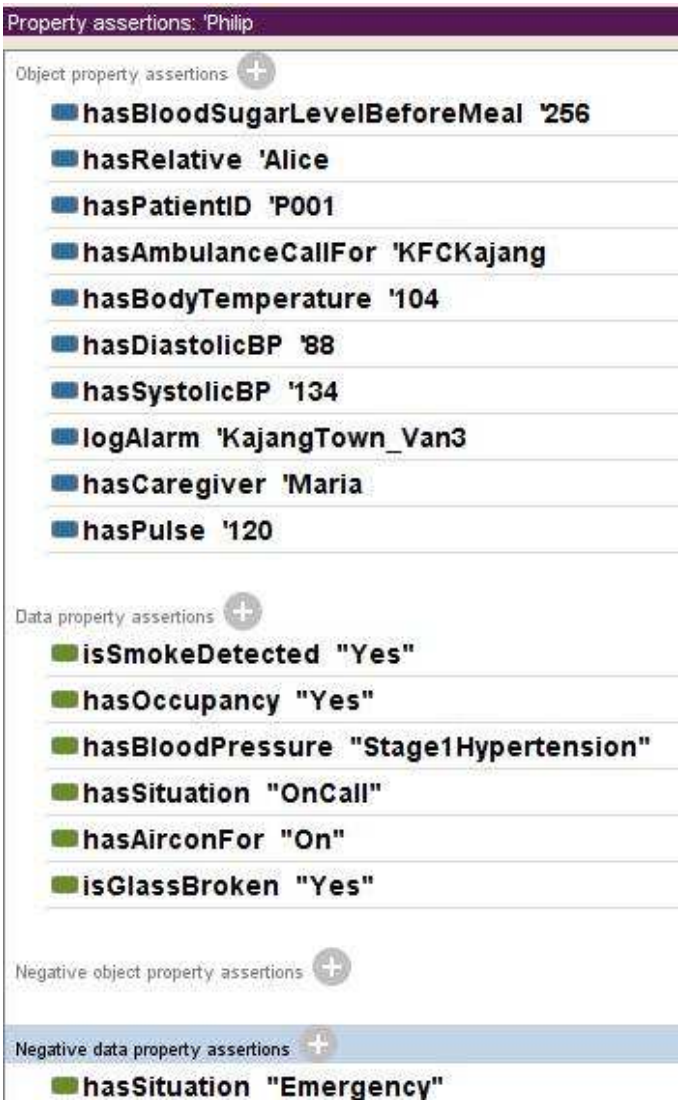

Fig. 3. An individualized smart environment ontology

The agents in Fig. 4 are designed using the translated Horn clause rules of the ontology. The translation process is automated which uses a java-based translator using OWL API [22] version 3.4.10. The OWL API is a high level application programming interface (API) that supports the creation and manipulation of OWL ontologies. It is a java based API for loading, saving, parsing and serializing ontologies in a variety of different syntaxes (defined in W3C specifications) such as RDF/XML, OWL/XML, functional syntax, etc. We choose Eclipse development framework to translate ontology axioms (OWL 2 RL and SWRL rules) into a set of plain text Horn-clause rules. This translator requires ontology IRI (International Resource Identifier) to

[1]https://dl.dropboxusercontent.com/u/34803034/OntologyAndSystemFiles.zip

access the elements from the ontology. Each ontology has an ontology IRI to identify ontology and their classes, properties and individuals. We use the OWL API 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 OWL API does not have a direct support for reading and writing ontologies in different syntaxes. The uses of parser and renderer in the reference implementation of OWL API make this task easier to customize ontologies in different syntaxes. When a specific parser is selected, ontologies are loaded and saved back in the same format from which it is parsed. The translator's core functions are:(1) System Prompt user to choose ontology, (2) Load ontology files (an output file of the Protégé editor [23]) as an input from link where it is published online, (3) Extract the set of logical axioms from the ontology, which can either be TBox axiom or ABox axiom, (4) We use OWL parser to parse ontology into OWL API objects to extract the set of TBox and ABox axioms, (5) The set of TBox axioms are already in the form of OWL 2 RL rules, we translate these set of axioms into a plain set of text in Horn clause rules format, (7) DL safe rule axioms are in the form of SWRL rules (in the ontology) which can be extracted to transform into plain text of Horn-clause rules format.

A. Specifying and verifying the system

We have considered three facets of the system while specifying and verifying its interesting properties using the Maude LTL model checker. This is partly because to observe and compare model checking performace and scalability. The first system is modelled using five agents, namely $1, 2, 3, 4$ and 5 which monitors the residents' (e.g., patient's) vital information such as Pulse Rate, Body Temperature, Blood Sugar Level etc. The system infers appropriate contexts based on the current contextual information of the patient whether e.g., there is an emergency situation or not, among others. In this system, the agents $2, 3, 4$ and 5 are able to infer high-level contexts from sensed low-level contexts using Horn clause rules in their knowledge-bases. These agents can classify current blood pressure, blood sugar, and pulse rate into different categories based on their current measurement values. E.g., agent 2's knowledge-base contains rules including the following:

Person(?p), SystolicBP(?sbp), DiastolicBP(?dbp), hasSystolicBP(?p, ?sbp), hasDiastolicBP(?p, ?dbp), greaterThan(?sbp,'120), lessThan(?sbp,'140), greaterThan(?dbp,'80), lessThan(?dbp,'90) $\rightarrow$ *hasBloodPressure(?p, 'Prehypertension) ;*

Person(?p), SystolicBP(?sbp), DiastolicBP(?dbp), hasSystolicBP(?p, ?sbp), hasDiastolicBP(?p, ?dbp), greaterThan(?sbp,'140), lessThan(?sbp,'160), greaterThan(?dbp,'90), lessThan(?dbp,'100) $\rightarrow$ *hasBloodPressure(?p, 'Stage1hypertension) ;*

hasBloodPressure(?p, 'Stage1hypertension) $\rightarrow$ *Tell(2,1, hasBloodPressure(?p, 'Stage1-hypertension)) .*

The first rule classifies that the person has blood pressure category *Prehypertension* if her Systolic Blood Pressure is greater than 120 and Diastolic Blood Pressure is greater than 80. That is, agent 2 may infer high-level context *hasBloodPressure('Philip, 'Prehypertension)* when the rule matches with the agent's working memory contexts, e.g., *Person('Philip), SystolicBP('134), DiastolicBP('88), hasSystolicBP('Philip, '134), hasDiastolicBP('Philip, '88), greaterThan('134,'120), lessThan('134,'140), greaterThan('88,'80), lessThan('88,'90),* and so on. The third rule is a communication rule of agent 2 through which it interacts with agent 1 and passes the context *hasBloodPressure('Philip, 'Prehypertension)* when it believes that *Philip* has *Prehypertension* at the moment. Similar to the above, agent 2 and all other agents in the system have other deduction and communication rules for other categories. Note that, the ontology driven rules do not have priority and a system designer is responsible to provide appropriate rule priorities while encoding the system into Maude, the complete set of translated rules for each of the three system designs can be found online[2]. In order to model this first scenario we have derived 105 Horn clause rules from the smart environment ontology and distributed them to the agents as working memory facts and knowledge base rules. E.g., the knowledge of agent 1 contains 45 rules, agent 2 is modeled using 10 rules, and so on. Whenever agent 1 receives most recently generated contexts from other agents, it infers current status of a patient and declares whether the patient has an emergency situation or not. The core inspiration is that each agent keeps the most recently derived contexts in the memory by overwriting an existing context, and this happens if agent's memory is full or a contradictory context arrives in the memory (even if the memory is not full). We verified a number of interesting resource-bounded properties of the system including the following non-conflicting contextual properties to see for example, when there is an emergency situation for a patient then the system should not produce non-emergency situation at the same time.

$$Prop1.1: \quad F(B_1\, hasSituation('Philip,' Emergency))$$

$$Prop1.2: \quad F(B_1\, Not(hasSituation('Philip,' Emergency)))$$

$$Prop1.3: \quad G(B_1 \sim (hasSituation('Philip,' Emergency) \wedge Not(hasSituation('Philip,' Emergency))))$$

The initial working memory facts (contexts) and rules are assigned to the agents in such a way that the system can infer both *hasSituation('Philip,'Emergency))* and *Not(hasSituation('Philip,'Emergency)))* contexts that are conflicting. The operator B_i used in the properties to state that agent i believes a context (in other words certain context appears in the agent i's working memory); and as usual G stands for always (globally), F stands for eventually (in the future), and X stands for next step. The truth of the first two properties ensure that indeed both these contexts can be inferred in the future, while the truth of the third property ensures that both of them never appear in the agent's memory at the same time. The above properties are verified as true and Maude reported the number of states explored and time required to verify them are 172 and $30ms$(milliseconds) for $Prop1.1$, 280 and $45ms$ for $Prop1.2$, and 2332 and $384ms$ for $Prop1.3$. While verifying these properties minimum memory space required by agent 1 was 12 units and it exchanged 4 messages.

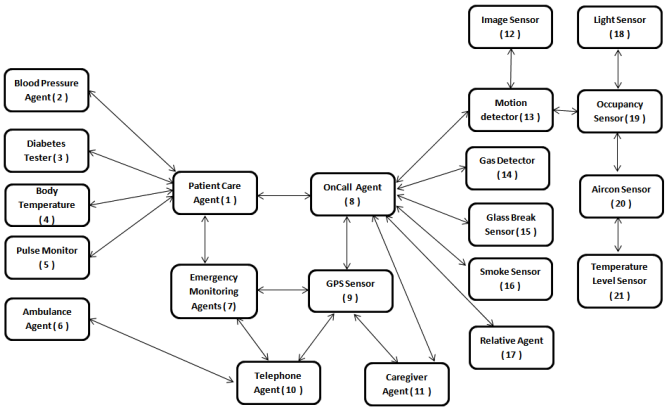

Fig. 4. Context-aware agents and their possible interactions

The second system that we consider for the verification is modeled using 11 agents, namely, $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ and 11, and to model this second scenario 133 Horn clause rules have been used. This system, in addition to inferring the residents' health status, interacts with various other agents to take appropriate actions. This enhances the services and making system more complex. For example, agent 1 interacts with Emergency monitoring agent and OnCall

[2]https://dl.dropboxusercontent.com/u/34803034/OntologyAndSystemFiles.zip

agent which in turn interact with various other agents to locate GPS coordinate points to call Ambulance via Telephone agents. Upon receiving message from agent 10, ambulance could move to GPS located point to rescue a patient. In addition, Caregiver is also notified by OnCall agent about the emergency situation with GPS coordinates point of the patient. We verified a number of interesting resource-bounded properties of this system including those we considered above in the first system.

$Prop2.1: \quad F(B_1\, hasSituation('Philip,' Emergency))$

$Prop2.2: \quad F(B_1\, Not(hasSituation('Philip,' Emergency)))$

$Prop2.3: \quad G(B_1 \sim (hasSituation('Philip,' Emergency) \wedge Not(hasSituation('Philip,' Emergency))))$

$Prop2.4: \quad G(B_8(Tell(1,8,hasSituation('Philip,' Oncall)) \wedge (Tell(9,8,hasGPSLocation('Philip,' KFCKajangTown)) \rightarrow X^n B_8\, Tell(8,11,hasAlarmFor('Philip,' KFCKajangTown))$

the fourth property above specifies that whenever agents 1 and 9 tell agent 8 that *Philip* has *OnCall* situation and his GPS location is at *KFCKajangTown*, within n time steps agent 8 sending an alarming message to agent 11. All the above properties are verified as true and Maude reported the number of states explored and time required to verify them are 282 and $60ms$ for $Prop2.1$, 640 and $131ms$ for $Prop2.2$, 4336 and $1048ms$ for $Prop2.3$, and 4336 and $1050ms$ for $Prop2.4$. While verifying these properties minimum memory space required by agent 1 was 14 and agent 8 by 8 units and the value of n was 4 (i.e., within 4 time steps agent 8 sending a alarming message to agent 11). The messages that the agents exchanged were: agent 1: 5, agent 2: 1, agent 3: 1, agent 4: 1, agent 5: 1, agent 6: 1, agent 7: 3, agent 8: 3, agent 9: 2, agent 10: 2, and agent 11: 1.

The third system that we consider for the verification is modeled using all the 21 agents, and to model this scenario 201 Horn clause rules have been used. This system models very complex scenarios and deals with a very high level of combinatorial aspects. It includes some smart home sensor agents to provide ease, comfort, security and healthy life in the smart home. In this system, the sensor agents (agents $12-21$) monitor the basic safety measures at home and inform to relatives of the patient for any kind of mishap occurrence in the smart home. For example, burglar alarm will ring in case e.g., smoke is detected by the Smoke sensor agent, and then OnCall agent immediately interact with the Relative agent to take appropriate actions. This system also checks the existence of a person in a room and automatically switch on/off the light and air-condition based on the current contexts. So saving energy is the additional requirement of the system. However, by adding more agents the system designer can make the system much more complex. We verified a number of interesting resource-bounded properties of this system including the following:

$Prop3.1: G(B_8(Tell(1,8,hasSituation('Philip,' Oncall)) \wedge (Tell(9,8,hasGPSLocation('Philip,' KFCKajangTown)) \rightarrow X^n B_8\, Tell(8,11,hasAlarmFor('Philip,' KFCKajangTown))$

$Prop3.2: G(B_{11}(Tell(8,11,hasAlarmFor('Philip,' KFCKajang-Town)) \rightarrow X^n B_{11}\, logAlarm('Alice,' Philip))$

the first property is same as $Prop2.4$ above, while second property above specifies that whenever agent 8 tells agent 11 that *Philip* has alarming situation and his GPS location is at *KFCKajangTown*, within n time steps *Alice* (caregiver agent 11) noticing this. Both the above properties are verified as true and Maude reported the number of states explored and time required to verify them are 379210 and $165461ms$ for $Prop3.1$, and 379210 and $164321ms$ for $Prop3.2$, and the value of n in $Prop3.2$ is 2. However, when we assign a value to n which is less than 4 in $Prop2.4$, and less than 2 in $Prop3.2$ the properties are verified as false and the model checker returns counterexamples. Similarly, when we assign a value to memory size (or message counter) which is less than the minimal required value, properties are verified as false. This also ensures the correctness of the encoding in that model checker does not return true for arbitrary values of n, memory and message counters. Note that, verification of true formulas take longer than verification of false formulas since a model checker will find a counterexample faster than it takes to explore the whole model.

VI. Related work

There has been considerable work in context-aware computing literature focusing on various domains including health care [24], [25], [3] just to mention but a few. Much of this work concentrate on representing and reasoning about contexts. However, unlike many other context-aware application systems, in many cases health care systems are considered as safety critical systems [2]. In such systems, not meeting design objectives may result in tremendous loss including possibly human lives. In [26], a formal system modeling framework is presented for analyzing pervasive computing systems, which covers various aspects of pervasive computing systems including context-awareness and concurrent communication. The authors have adopted CSP like hierarchical modeling language to model desired systems, which could be used to encoded and verify system properties using existing model checking techniques. Their proposed approach has been demonstrated by modeling a health care case study and desired properties of the system have been verified using the PAT model checker [27]. However, it is not clear whether the predefined rules written in Drools are considered in the verification process and how those rules are encoded while verifying the system properties. In [28], the Adaptation Finite-State-Machine (A-FSM) is proposed for modeling and verifying context-aware adaptive behavior of mobile applications. The authors have proposed some algorithms that are used to automatically detect fault patterns based on the A-FSM. A-FSM detects adaptation faults by exhaustively exploring the space constructed by all possible value assignments to context variables used in a Context-Aware Adaptive Application's (CAAAs) rules. In [29], authors have used SPIN model checker to model a Smart Home environment and discussed consistency in context-aware behavior. They have emphasized the good rule design practices based on general observations, and shown how rules of a context-aware system could be inconsistent and may lead to undesired system behavior. In [30], authors have proposed a context-aware design and verification framework based on Role-Oriented Adaptive Design (ROAD) [31], and they have used ROAD4Context approach to model context-aware interactions, abstract processes and invariant properties of the desired systems. The functional behavior of the desired system is modeled using UML-based process models, which are further translated into Petri nets-based formal models, while invariant properties of the systems are expressed as LTL formulas. Compared to these modeling and verification approaches, our proposed $\mathcal{L}_{\mathcal{DROCS}}$ provides formal framework to model structural and behavioral aspects of context-aware systems considering their resource-bounded features and how such models can be encoded and verified using the Maude LTL model checker. In [11], it has been shown how context-aware systems can be modeled as resource-bounded rule-based systems using ontologies, however it is based on monotonic reasoning where beliefs of an agent cannot be revised based on some contradictory evidence. In [32], [33] authors have presented automated verification of resource requirements of reasoning agents using the Mocha model checker. In [34] the same authors presented preliminary work considering first order Horn clause rules and Maude LTL model checker, and illustrated the scalability of their approach by comparing it to results presented in [32]. In [35] authors presented framework to verify heterogeneous multi-agent programs based on meta-APL, where a heterogeneous multi-agent program is initially translated to meta-APL and then resulting system is verified using Maude. In this work, we specify and verify $\mathcal{L}_{\mathcal{DROCS}}$ models using Maude because it can model check systems whose states involve arbitrary algebraic data types. Furthermore, unlike [33], Rule variables can be represented directly in the Maude encoding, without having to generate all ground

instances resulting from possible variable substitutions.

VII. Conclusions and future work

In this paper, we show how a $\mathcal{L}_{\mathcal{DROCS}}$ [5] model can be encoded using the Maude LTL model checker [6] and its interesting properties can be verified automatically. Using an example system, we show how to analyze and verify non-conflicting context information guarantees it provides. In future work, we plan to extend $\mathcal{L}_{\mathcal{DROCS}}$ logical framework to specify and verify heterogeneous multi-context systems, which will allow different reasoning techniques (logics) to be used for different agents and contextual information flow among agents will be modeled via bridge rules.

References

[1] Bricon-Souf, N., Newman, C.R.: Context awareness in health care: A review. International Journal of Medical Informatics 76(1), 2–12 (2007)

[2] Bardram, J.E., , Nørskov, N.: A context-aware patient safety system for the operating room. In: Proceedings of of the 10th international conference on Ubiquitous computing. pp. 272–281 (2008)

[3] Esposito, A., Tarricone, L., Zappatore, M., Catarinucci, L., Colella, R.: A framework for context-aware home-health monitoring. LNCS, vol. 5061, pp. 119–130. Springer (2008)

[4] Byun, H.E., Chevers, K.: Utilizing context history to provide dynamic adaptations. Applied Artificial Intelligence 18(6), 533–548 (2004)

[5] Rakib, A., Haque, H.M.U.: A logic for context-aware non-monotonic reasoning agents. In: Proceedings of the 13th Mexican International Conference on Artificial Intelligence, MICAI 2014. Lecture Notes in Computer Science, vol. 8856, pp. 453–471. Springer (2014)

[6] Eker, S., Meseguer, J., Sridharanarayanan, A.: The maude LTL model checker and its implementation. In: Ball, T., Rajamani, S.K. (eds.) SPIN2003. LNCS, vol. 2648, pp. 230–234. Springer-Verlag (2003)

[7] Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: Proceedings of the First Workshop on Mobile Computing Systems and Applications. pp. 85–90. IEEE Computer Society, Washington, DC, USA (1994)

[8] Brown, P.J., Bovey, J.D., Chen, X.: Context-aware applications: from the laboratory to the marketplace. IEEE Personal Communications 4(5), 58–64 (1997)

[9] Dey, A., Abwowd, G.: Towards a better understanding of context and context-awareness. Technical Report GIT-GVU-99-22, Georgia Institute of Technology (1999)

[10] Daniele, L., Costa, P.D., Pires, L.F.: Towards a rule-based approach for context-aware applications. In: EUNICE. Lecture Notes in Computer Science, vol. 4606, pp. 33–43. Springer (2007)

[11] Rakib, A., Haque, H.M.U., Faruqui, R.U.: A temporal description logic for resource-bounded rule-based context-aware agents. In: Context-Aware Systems and Applications. LNICST, vol. 128, pp. 3–14. Springer (2014)

[12] Patel-Schneider, P..F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation, World Wide Web Consortium, 10 February, (2004)

[13] ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. Web Semantics: Science, Services and Agents on the World Wide Web 3(2-3), 79–115 (2005)

[14] Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: WWW2003. pp. 48–57. ACM Press (2003)

[15] 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)

[16] Pollock, J.L.: Defeasible reasoning. Cognitive Science 11(4), 481–518 (1987)

[17] Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Transactions on Computational Logic 2(2), 255–287 (2001)

[18] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J.: All About Maude - A High Performance Logical Framework, Lecture Notes in Computer Science, vol. 4350. Springer-Verlag (2007)

[19] Bikakis, A., Antoniou, G., Hasapis, P.: Strategies for contextual reasoning with conflicts in ambient intelligence. Knowledge and Information Systems 27(1), 45–84 (2011)

[20] Leijdekkers, P., Gay, V.: Personal heart monitoring system using smart phones to detect life threatening arrhythmias. In: Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on. pp. 157–164. IEEE (2006)

[21] Moreno, A.: Medical applications of multi-agent systems. Computer Science & Mathematics Department, Universitat Rovira i Virgili ETSE. (2007)

[22] 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)

[23] Protégé: The Protégé ontology editor and knowledge-base framework (Version 4.1). http://protege.stanford.edu/ (July 2011)

[24] Bardram, J.E.: Hospitals of the future ubiquitous computing support for medical work in hospitals. In: Proceedings of UbiHealth'03 the 2nd international workshop on ubiquitous computing for pervasive healthcare applications (2003)

[25] Paganelli, F., Giuli, D.: An ontology-based context model for home health monitoring and alerting in chronic patient care networks. In: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops, 2007, pp. 838–845 (2007)

[26] Liu, Y., Zhang X., Dong J.S., Liu Y., Sun J., Biswas J., and Mokhtari M.: Formal analysis of pervasive computing systems. In: 17th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 169-178. IEEE, (2012)

[27] Sun, J., Liu, Y., Dong, J.S., and Pang, J.: PAT: Towards Flexible Verification under Fairness. In: Proceedings of the 21th International Conference on Computer Aided Verification (CAV'09), LNCS 5643, pp. 709-714 (2009)

[28] Sama, M., Elbaum, S., Raimondi, F., Rosenblum, D. S., and Wang, Z.: Context-aware adaptive applications: Fault patterns and their automated identification. In: IEEE Transactions on Software Engineering, vol. 36, no. 5, pp. 644-661, (2010)

[29] Preuveneers, D., Berbers, Y.: Consistency in context-aware behavior: a model checking approach. In: Intelligent environments (workshops), ambient intelligence and smart environments, vol 13, IOS Press, pp 401412 (2012)

[30] Tran, M. H., Colman, A., Han, J., and Zhang, H.: Modeling and verification of context-aware systems. In: 19th Asia-Pacific Software Engineering Conference (APSEC), vol. 1, pp. 79-84. IEEE, (2012)

[31] Colman, A.: Role-Oriented Adaptive Design, PhD Thesis,Faculty of Information and Communication Technologies, Melbourne: Swinburne University of Technology (2006)

[32] 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.) MoChArt2008. LNCS, vol. 5348, pp. 1–14. Springer,Heidelberg (2008)

[33] Alechina, N., Jago, M., Logan, B.: Modal logics for communicating rule-based agents. In: Proceedings of the 17th European Conference on Artificial Intelligence A. pp. 322–326 (2006)

[34] Alechina, N., Logan, B., Nga, N.H., Rakib, A.: Automated Verification of Resource Requirements in Multi-Agent Systems Using Abstraction. In: van der Meyden, R., and Smaus, J. G., (eds.) MoChArt2010. LNCS, vol. 6572, pp. 69-84. Springer,Heidelberg (2010)

[35] Doan, T.T., Yao, Y., Alechina, N., Logan, B.: Verifying heterogeneous multi-agent programs. In: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems. pp. 149–156. International Foundation for Autonomous Agents and Multiagent Systems (2014)