**Noname manuscript No.**
(will be inserted by the editor)

# A framework for implementing formally verified resource-bounded smart space systems

**Ijaz Uddin · Abdur Rakib · Hafiz Mahfooz Ul Haque**

Received: date / Accepted: date

**Abstract** Context-aware computing is a mobile computing paradigm that helps designing and implementing next generation smart applications, where personalized devices interact with users in smart environments. Development of such applications is inherently complex due to these applications adapt to changing contextual information and they often run on resource-bounded devices. Most of the existing context-aware development frameworks are centralized, adopt client–server architecture, and do not consider resource limitations of context-aware devices. This paper presents a systematic framework to modelling and implementation of resource-bounded multi-agent context-aware systems on Android devices. The proposed framework makes use of semantic technologies for context modelling and reasoning about resource-bounded context-aware agents, Android powered smartphones as development platform, a suitable communication model and declarative rule-based programming as a preferred development language.

**Keywords** Context-awareness · Resource-bounded agents · Rule-based reasoning · Non-monotonic reasoning · Android SDK

## 1 Introduction

The last few decades have seen an exponential growth and change in computing technologies. Computers have evolved from big bulky mechanical machines into lightweight lightning fast laptops and tablets. While computers were successfully prospering, there was the beginning of the mobile phones. In 1973 Motorola first introduced hand held telephone device [1]. It was not until 1980 that the use was slowly transferring to public use. The late 20th century has witnessed the transfer of mobile phone into smartphone. Smartphones are now capable to carry out our daily routine tasks, which were earlier possible on computers or other similar devices only, such as browsing Internet, social networking, taking photos or making videos and so on [2]. With the advancements of the smartphone combined with feature-rich softwares, applications and Internet connectivity make it more easier for people to share their experiences using social networking applications, including VoIP services, free messaging and call applications, to name some [3]. Along with variuos high-tech features, a smartphone is also equipped with a wide range of sensors, including global positioning system (GPS), shake sensors, accelerometers, and proximity sensors [4]. These sensors that accommodate a user in his daily life can further be used in a large variety of applications, which can provide user related and surrounding information as *contexts*. These sensors can be integrated in a way to provide enough user information, including user's location, time, movement, and surrounding environmental information. When provided with a suitable communication mechanism, it can also enhance interaction between the user, application and other devices [5]. The smartphones or other devices that are used to implement such applications may act as *intelligent agents* for a particular scenario of an application. Thus smartphones and agent-based technology can provide tremendous benefits for the development of context-aware mobile applications.

In the literature various definitions of *context* exist (see e.g., [6,7]). Dey et al. define context as 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-aware system is a system which uses context to provide relevant information and/or services to its user based on the user's

Ijaz Uddin, Abdur Rakib, and Hafiz Mahfooz Ul Haque
School of Computer Science, The University of Nottingham, Malaysia Campus
E-mail: {khyx4iui,Abdur.Rakib,khyx2hma}@nottingham.edu.my

tasks [7]. The context-aware computing paradigm emerged in early 1990s with the introduction of small mobile devices. In 1992, Olivetti Labs active badges used the infrared badger assigned to staff members for tracing their locations in office and according to the locations calls were forwarded [8]. Further developments in the field lead to the development of various context-aware frameworks such as Georgia Tech's Context Toolkit [9]. In recent years, more research has been carried out and advanced context-aware systems exist [10], and the contributions to research and development over the years promise a bright future of such systems. Generally, context-aware systems interact with human users, they often exhibit complex adaptive behaviours, they are highly decentralised and can naturally be implemented as multi-agent systems. An agent is a piece of software that requires to be reactive, pro-active, and that is capable of autonomous action in its environment to meet its design objectives [11]. 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 these non-human agents is implementing them as rule-based reasoning agents. Rule-based reasoning and traditional rule engines have found significant applications in practice, though mostly for desktop environment where the resources (computational and communication) are abundantly available compared to smartphone devices. The main issue with those rule engines is that they cannot be easily used on smartphones or resource bounded devices due to platform differences and different hardware profiles. Some rule engines, which are discussed in Section 2, have already been tested for porting into mobile environment but the results were either not satisfactory or the porting were only partially successful. In view of the above, there is a need to develop a decentralized formal context-aware computing model that makes use of the smartphone platform, a suitable communication model and declarative rule-based programming as a preferred development language. This paper addresses some of these issues by exploring the practical implementation of the framework presented in [12]. By developing a pure smartphone compatible context-aware systems development framework, any kind of domain specific context-aware applications can be developed, e.g., elder care system, hospital critical situation, traffic control and office security, among others.

The remainder of the paper is organized as follows. In Section 2, we briefly introduce context-aware computing and its limitations and challenges in resource-constrained settings, followed by a discussion of a formal context modelling and reasoning framework [12]. Section 3 presents rule-based context-aware system specification and how we implement the logical framework developed by [12]. Section 4 discusses detailed implementation and internal working mechanisms of the rule-based context-aware systems. Section 5 presents implementation of a prototype example system considering three agents in a smart environment. Section 6 concludes and discusses some suggestions for future work.

## 2 Background Study

The rapid development of mobile technologies and new research in pervasive computing have sparked a renewed interest in context-aware applications. A large part of research on context-aware systems and applications studies formal approaches to modelling context and reasoning techniques for contextual information. Reasoning techniques help to realise the adaptation of an application to the changing environment and also to infer higher level contextual information from sensed or available low level contextual information. In the literature, various context modelling and reasoning approaches have been proposed, including ontology and rule-based approach [13–17, 12]. The work by [14] proposes ontology based context management (GCoM) model to address context modelling and reasoning. Where the rules used in the reasoning process can be user defined and/or ontological. It shows the methodology from context acquisition till expression in resource saving manner and the model has the capacity for re-usability. The work by [15] on the other hand focuses on the rapid prototyping of context-aware application development. The emphasis is given to the shared conceptualization of the domain being a collaborative environment. The users are divided into three wide categories based on their skill set into high, middle and low level. Based on the user skills, the user is provided some privileges in its environment. The framework, besides other components, uses resource sharing server which suggests limitation on distributed approach towards the framework. In our previous work [12, 17], we have shown that ontological and logic based approach is a good way for modelling context-aware systems, and it allows us to model context-aware systems as rule-based reasoning agents. A logic defines the conditions in which a concluding fact may be derived. In [12], we developed a logical framework for resource-bounded context-aware multi-agent systems which handles inconsistent context information using non-monotonic reasoning, however, the framework was not implemented using smart devices. To the best of our knowledge and the study backed by the work of [18] there does not exist any concrete framework for mobile platform that addresses the issues of context-aware system development considering mobile device resource constraints. A framework that may cover all the aspects of context-aware applications in mobile platforms, including methodology, language, inference engine and communication mechanisms. Development of such a framework may be beneficial both to the developers and researchers [18]. Some attempts have already been made to port the existing desktop frameworks into the mobile platform but the results were not satisfactory, mainly due to the platform and hardware differences [19,20], where the task was to port the JADE framework into Android environment. Similarly the work by [21], which is based on Android, uses context expressions. A context expression is a Boolean expression, in which axioms are the context condition on the context entities. Although the work is based on Android, the framework doesn't have its own language. Furthermore, instead of reasoning, various scenarios are monitored using the evaluators (==, >=, >, <, <==, regular expression, distance). The authors intend to provide distributed environment compatibility in their future work. From the programming languages perspective, there are some tools like Kivy framework that can be used for programming Android Applications. However, when we need the latest updates and especially as in our case access to the sensors and their support, using these tools may not be a good option [22].

Therefore, although various context-aware frameworks have been developed over the years, their functions remain primitive. This is because these systems are more complex due to the mechanism to sense and reason about contextual information and their changes over time. Furthermore, such systems often run on tiny resource-bounded devices in highly dynamic environments. Many challenges might arise when these context-aware devices perform computation to infer implicit contexts from a set of given explicit contexts and reasoning rules, and perhaps exchange information via messages. In the following section, we list some constraints that often arise while designing and developing context-aware mobile applications.

## 2.1 Limitations and Challenges in Resource Bounded Devices

The insight study of the resource-bounded devices suggests that most of the widely used platforms, such as iOS, Android, Windows mobile, and blackberryOS, share common limitations. Some major limitations are listed below which a developer faces when developing context-aware mobile applications.

- **Processing power** The processing power of a standard computer is very adequate to run multiple programs simultaneously. It has multiple processors, cores and a variety of supporting hardware that can be further installed to make the processing faster and smoother. On the other hand, a smartphone is limited to what is offered by the manufacturer and there is no possibility to add or enhance it by adding any hardware. So, practically the processing power of a smartphone is comparatively much weaker than a desktop computer. Also, a developer should keep in mind that the application should not engage the processor more than required. Excessive use of a processor may slow down the overall performance of the smartphone and can cause the battery to drain faster.
- **Memory limitations** Contrary to the desktop computers which offer a huge amount of storage in TeraBytes and RAM in GigaBytes, the smartphones still use GigaBytes of storage space which are normally ranges from 8GB to 128GB. While the RAM is 4GB on various latest smartphones, perhaps more RAM expected in coming years, which is still less than desktop systems. The applications for desktop has vast amount of memory available both in terms of storage and RAM to operate at optimum pace even when other softwares are running simultaneously. However, resource-bounded devices need more memory optimization to make the application run smoothly, the scenario can get worse when there are multiple applications running.
- **Device size** In smartphones, the size is a factor, compromises are made on certain hardware to keep the size comfortable and not bulky. As stated earlier, that in smaprtphones a user is bound to use what is offered by a manufacturing company. A user can not add more RAM or Internal memory. Developers have to design the applications keeping in view of the hardware resources available, as there is no option to make hardware changes. Unlike desktop computers, in smartphones almost every thing comes embedded, besides few options such as battery, SD-card etc.
- **Battery power** The smartphones are made for mobility. This mobility is powered by the battery. A smartphone when connected with a power source has virtually unlimited battery life but no mobility at all [23]. To keep the balance between mobility and power, it is very crucial to use the resources in energy saving manner to maximize the battery operating time. This includes application development in such a way that it should not engage the resources continuously. As doing so may cause the draining of the battery. Moreover, the communication between devices or agents also consumes energy. Keeping the communication optimized ultimately helps in prolonging the battery life.
- **Programming language** Using desktop computers a developer can choose any language to program application softwares, including C, C++, C#, PHP, Java, Python and many more. However, on smartphones a developer is restricted to the use of the programming language which is compatible with the platform of a smartphone and the developer has very little choice in this regard.

Due to the above mentioned issues it is not desirable to directly port any software to the smartphone platforms, instead it has to be recoded according to a chosen platform. Furthermore, the constraints mentioned above can have a great impact on the development, specifically due to the bounded resources and limitations of the smartphone platforms.

## 2.2 A Context Modelling and Reasoning Framework

In our study, we realize 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 [7]. 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. For example, we can characterize user's current status of a context-aware system based on the contexts *"Mary has blood pressure categorized as High"* as $\langle Mary, hasBloodPressure, High\rangle$ or using first order logic term as $hasBloodPressure(Mary, High)$. In [12], we studied ontology-based context modelling approach and for that purpose we use OWL 2 RL, a profile of the new standardization OWL 2, and based on $pD^*$ [24] and the description logic program (DLP) [25]. 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 the DLP technique [25]. Furthermore, we express more complex rule-based concepts using SWRL [26] which allow us to write rules using OWL concepts.

In our conceptual and logical framework [12], we consider context-aware agents having constraint on various resources, namely time, memory, and communication. Each agent's memory usage is modelled as the maximal number of contexts to be stored in the agent's working 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. Here, we briefly describe the notion of contexts and context-aware reasoning that is used in the logical model. Each agent $i \in A_g$ in a multi-agent reasoning system has a program, consisting of a finite set of strict and defeasible rules (these are essentially Horn clause rules), and a working memory, which contains facts (current contexts). If an agent $i$ has a rule:

*Patient(?p), hasBloodPressure(?p,High), hasGPSLocation(?p, ?loc)* $\rightarrow$ *hasAlarmingSituation(?p, ?loc)*

and the contexts *Patient(Mary)*, *hasBloodPressure(Mary, High)* and *hasGPSLocation(Mary, UNMC)* are in the agent's working memory and *hasAlarmingSituation(Mary, UNMC)* is not in the agent's working memory in state $s$, then the agent can fire the rule which adds the context *hasAlarmingSituation(Mary, UNMC)* 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 working memory (even if the working memory is not full). We say that two contexts are contradictory iff they are complementary with respect to $\sim$, for example, *hasAlarmingSituation(Mary, UNMC)* and $\sim$*hasAlarmingSituation(Mary, UNMC)* are contradictory contexts. As we use defeasible reasoning to model a system, conflicting context can be represented using $\sim$ in the working memory of an agent $i$. However, in practice conflicting contexts can be manipulated in different ways. During execution of the system, conflicting contexts can have different notions in the working memory of an agent $i$, for example, a conflicting context can be represented using the following notion: *hasAlarmingSituation(Mary, UNMC)* conflicts both with $\sim$*hasAlarmingSituation(Mary, UNMC)* and *hasAlarmingSituation(Mary, TTS)*, where *UNMC* and *TTS* represent distinct locations. Similarly, a conflicting context can also be of the form: *hasTemperature(Livingroom, High)* and *hasTemperature(Livingroom, Cool)*. Whenever a newly derived context arrives in the agent's 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 the above case *hasAlarmingSituation(Mary, UNMC)* will be a contradictory context if $\sim$*hasAlarmingSituation(Mary, UNMC)* is present in the agent's working memory, so $\sim$ *hasAlarmingSituation(Mary, UNMC)* will be replaced by the newly inferred context *hasAlarmingSituation(Mary, UNMC)*. In addition to firing rules, agents can exchange messages regarding their current contexts. A more detailed explanation can be found in [12,27]. In this paper, we extend our theoretical work presented in [12] by implementing the ontology and logic based framework using the Google Android SDK and smartphones, where smart devices (and hence agents) sense the surrounding environments to acquire low level contexts and infer high level contexts based on the rules that are derived from smart environment ontologies, communicate with each other, and adapt their behaviour accordingly.

## 2.3 Implementation Platform

To implement the above discussed framework first we need to select a suitable platform. Existing rule-based programming environments, such as JADE, JARE, JESS [28] and many more are written in Java. Java adds platform independence besides other rich libraries implementation. Since

Java is platform independent, in principle the systems developed in Java for one platform should work fine on any other platforms. However, this is not the case when we talk about implementation in resource-bounded environment especially for Android platform. The Java language that is used for the desktop programming, now known as Oracle Java is wide and a vast language. The point where the Java for Android differs from the Oracle Java is the less number of libraries support e.g., Swing is not supported in Android platform. The Android mainly supports Java core programming. Another big difference lies in the low level machine translation mechanism. In Java JVM or Java Virtual Machine is used to translate the code into platform specific code, while in Android instead of JVM, DVM or Dalvik Virtual Machine is used. The DVM of Android is a compact Virtual Machine that is used to run programs on resource-bounded devices [29]. The package of the Android application is installed from the apk file format which has internal difference with the jar file format as used in the typical Java programs. In terms of the usage Android has the major user base as of 2014, and in latest 2015 report [30]. In this work, we chose the Google Android SDK to implement resource-bounded context-aware applications, however this choice does not restrict the research objective to Android only, and in the future we aim to develop a context-aware implementation framework that can be used to run application programs on multiple platforms seamlessly.

## 3 Rule-Based Context-Aware System Specification

In this section, we explain different aspects of a rule-based reasoning system and its various components (see Figure 1). Where necessary, we have provided the flow charts and algorithms for better understanding of the system.

### 3.1 Architecture and Basic Hardware Usage

For the development of an example resource-bounded context-aware system, rule-based agents being developed in this paper are composed of multiple Android smartphones. For this purpose we used three different Android phones having different specifications. One phone has been rooted, formatted and installed with the custom ROM with Nokia x2 Hardware profile. Other two phones are Lenovo 1000 and 6600 respectively, having different specifications to check the application behaviour. A brief comparison is provided in Table 1.

### 3.2 Application Interface

The interface for our applications is provided in two different versions. While one is web based desktop environment, the other one is Android application based. In general the interface serves the purpose to add rules to the rule-base along with their priorities, initial facts (i.e., existing high-level contexts), and associated flags. The rules are checked for its validity before adding into the rule-base. Similarly the facts are also checked for their validity as to whether they follow the intended format. The priority as the name indicates for a rule give it preference over other rules. That is, a higher priority for a rule give it preference over other matching rules. The flag associated with every rule is used to specify the type of the rule. For instance, the character 'G' is used to represent a rule containing a Goal statement, which indicates that a certain rule execution results in goal achievement. The character 'C' represents the communication rules, which can trigger a communication between agents (devices). The character 'D' represents the deduction rules. The communication between agents are explained in more detail in Section 4.8.

#### *3.2.1 Desktop Interface*

The desktop interface is a web based application, which uses Apache web server, MySQL Database, Jquery and PHP language as its main components besides HTML for the interface design. The application is platform independent and runs in any standard browser on different platforms e.g Windows, Linux, Macintosh with a minimum setup. This interface can be made online to be accessed from any computer with Internet connection. The user can add rules from this interface, it allows validating both the Left Hand Side(LHS) or body and Right Hand Side(RHS) or head (or consequent) of any rule provided. If the rule qualifies the format specified, it will be returned as valid, otherwise user will be prompted to enter a rule according to the intended format. A rule has the following format:

$$m : P_1, P_2, \ldots, P_n \rightarrow P_0 \quad \text{where } n \geq 0.$$

where $m$ is the rule priority. Each $P_i$ is an atomic formula of the form $p(t_1, t_2)$, $Ask(i, j, p(t_1, t_2))$ or $Tell(i, j, p(t_1, t_2))$, where $i$ and $j$ ($i \neq j$) represent agents, $p$ is a predicate symbol and the $t_k$ are terms. Where $Ask$ and $Tell$ are special atoms used for communication between the agents [12]. Each term is either a constant symbol or a variable. Every variable occurring in a rule is universally quantified, and its scope is the clause in which the variable occurs. Every variable appearing in the head must also appear in the body of a rule. The "$\rightarrow$" is read as *if* and "," as *and*. The atom $P_0$ is called consequent (or head) of the rule and the conjunction $P_1, P_2, \ldots, P_n$ is the body of the rule. If $n = 0$, then the body is equivalent to TRUE and is called a fact otherwise its a rule.

The interface allows to create an agent program by receiving a set of rules and initial working memory facts. The various phases of the desktop interface are provided in Fig-

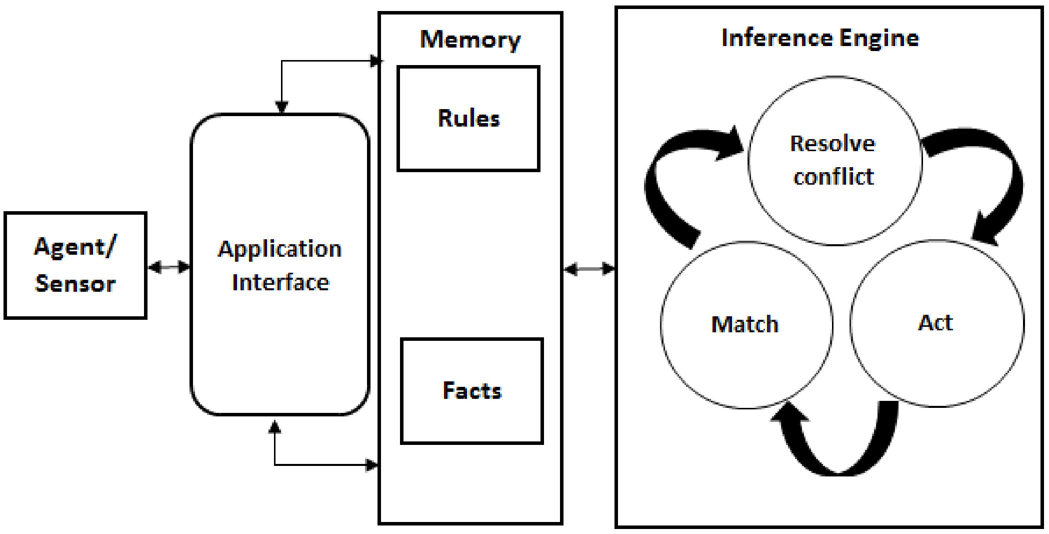
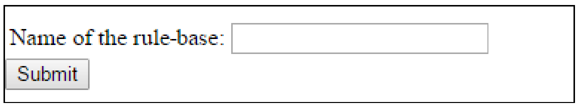


Fig. 1: System overview from an individual agent's perspective

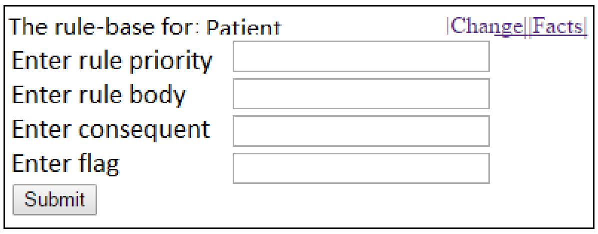
| Device | RAM | Processor | Internal Memory | OS | Embedded Sensors |
|---|---|---|---|---|---|
| Nokia X2 | 1 GB | Dual-core 1.2 GHz Cortex-A7 | 4 GB | Custom ROM | Accelerometer, proximity, and GPS |
| Lenovo A1000 | 1 GB | Quad-core 1.3 GHz Cortex-A7 | 8 GB | Android OS | Accelerometer and GPS |
| Lenovo 6600 | 1 GB | Quad-Core 1.5 GHz | 8 GB | Android OS | Accelerometer, proximity, and GPS |

Table 1: Smartphone comparison sheet

ures 2, 3, and 4. In Figure 2 (a), a system developer can create a rule-base. Once a rule-base is created the next interface shown in Figure 2(b), is ready to receive input a set of rules. Figure 3 shows the validation of the rule and allowing the system developer to save it if the rule entered is according to the correct format. Figure 4 shows the phase where the system developer wants to enter the facts, it auto suggest the rules as a developer starts typing so that the chances for adding irrelevant or erroneous facts are minimized. The Desktop interface produces its output in the form of a JSON [1] file. JSON is a light weight data interchange format. The JSON file is then further provided as an input to the Android Application.


Name of the rule-base:

(a) Creating an agent's rule-base


The rule-base for: Patient |Change||Facts|
Enter rule priority
Enter rule body
Enter consequent
Enter flag

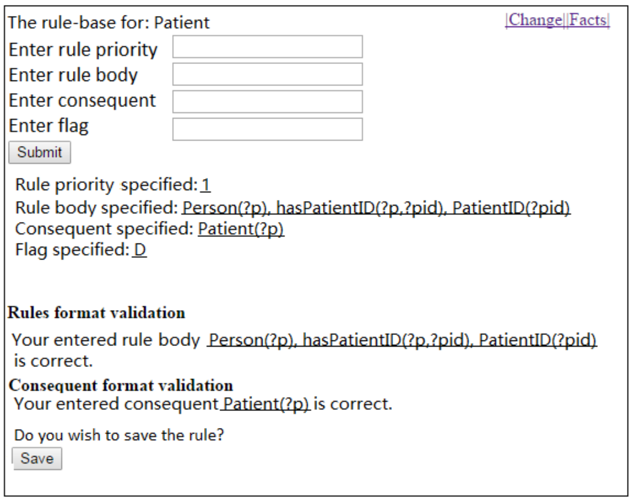
(b) Rules insertion interface

Fig. 2: Rule-base initialization

### 3.2.2 Android Interface

Figure 5 depicts Android interface to store the rules in case a developer wants to use the Android interface. We have options to insert rule priority, rule body, consequent of the rule, and the flag. The inserted rules are stored in the Android SQLite database. In the Android environment, SQLite is used for database operation without the need of any external application installation. The *SAVE* button saves the rule into the rule-base. The *INITIALIZE RULE-BASE* button takes the set of rules for processing to the next stage. The *CLEAR RULE-BASE* button is used to erase all the rules and related data from its rule-base. Figure 6 depicts how a de-

[1] JSON-http://www.json.org/


The rule-base for: Patient
|Change||Facts|
Enter rule priority
Enter rule body
Enter consequent
Enter flag
Submit

Rule priority specified: 1
Rule body specified: Person(?p), hasPatientID(?p,?pid), PatientID(?pid)
Consequent specified: Patient(?p)
Flag specified: D

**Rules format validation**
Your entered rule body Person(?p), hasPatientID(?p,?pid), PatientID(?pid) is correct.
**Consequent format validation**
Your entered consequent Patient(?p) is correct.
Do you wish to save the rule?
Save

Fig. 3: Validation of Horn-clause rules

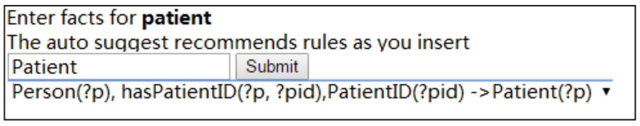

Enter facts for **patient**
The auto suggest recommends rules as you insert
Patient Submit
Person(?p), hasPatientID(?p, ?pid),PatientID(?pid) ->Patient(?p)

Fig. 4: Facts interface with auto suggestion

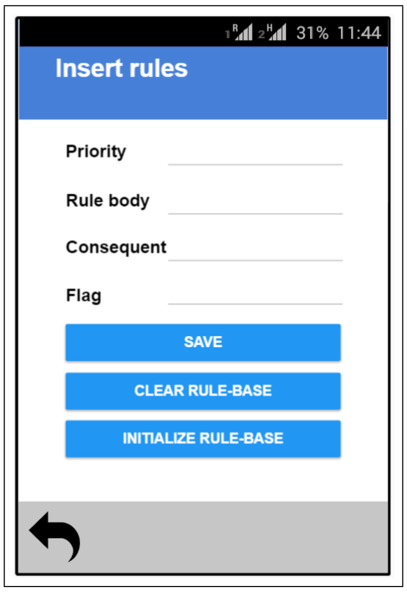
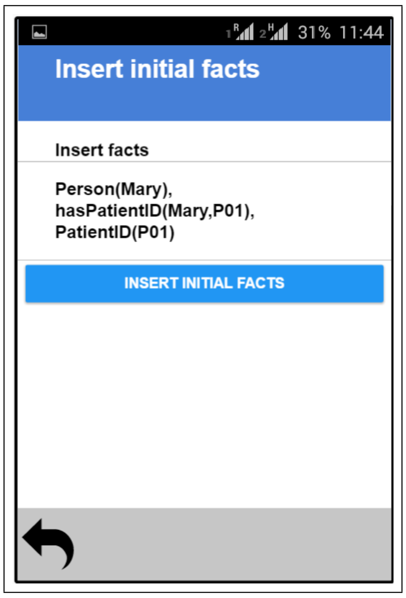


Fig. 5: Rule manipulation activity interface



Person(Mary),
hasPatientID(Mary,P01),
PatientID(P01)

Fig. 6: Initial working memory facts

veloper can use Android interface to enter the initial facts to start the rule-base. It is pertinent to enter the initial facts with care to make the system process them as intended. Alternatively user can also provide the initial facts from the JSON file generated from the desktop interface. In the next section, we will discuss in more detail how the backend works.

## 4 Internal Working Mechanism

The proposed system, as stated before, has different modules which are integrated with each other to perform the whole task. In this section, we explain the internal working mechanism of the rule-based agents. We explain each process from inserting the rules into the knowledge base till a Goal is achieved. We would also like to mention that unlike many other systems, our inference engine do not uses RETE algorithm [31]. The reason for not using the RETE algorithm is that it is very memory consuming algorithm due to its heavy use of beta memory and in resource-bounded environment memory is one of the bounded resources. Also our system design is based on decentralised approach, so rules of a smart space system are distributed among the agents, and often fewer rules are required to design most agents' behaviours. Therefore, instead of using RETE algorithm we developed our own pattern matching mechanism which is tailored to fit our requirements and resources.

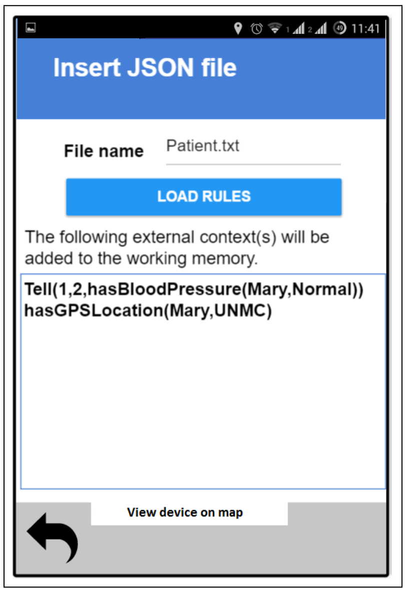
### 4.1 Usage of Key Terms

In this paper, we use some general terminologies which are defined and summarised here for better understanding. Agent refers to any device which is able to exhibit goal-directed intelligent behaviour and communicate with other agents and human users. Working memory is a short term memory where newly derived facts are stored, in our system it is limited and the agents are designed in a manner to work within the limitations provided. Rule-base is a knowledge-base of an agent. Flag refers to a characters associated with a rule with its own meaning e.g., 'G' for goal, 'C' for communication rule and

‘D’ for deduction. Priority is defined with each rule to give it priority over other rules, it is a positive numeric value. Activity is a term used to define the Android active screen on the display. For the rest of the paper we use the JSON files generated by the desktop interface, and all the subsequent operations are carried out accordingly.

### 4.2 System Architecture and Sensor Data Acquisition

The architecture of our system consists of three layers. In the first layer, environmental data is sensed either from the mobile device embedded sensors or independent sensors connected to a mobile device. In the second layer, the low-level contexts are generated from sensed data and then high-level contexts are inferred by the reasoning techniques. In the third layer, the context-aware application provides services based on the available contexts. Sensors are an important part of context-aware systems. They are used to collect environmental data as low-level contexts and forward those data to the intended device or agent, the agent then inferences high-level contexts through reasoning. In our system design we consider two types of sensors, first type which are embedded directly to the agent and the second type when the sensors are independent and are attached to different parts of a human body or in the surrounding environment. In the case where the sensors are embedded to an agent, for example, GPS on Android device, the data can be acquired using the GPS sensor API provided by the Android SDK. The figures 7 and 8 show how a sensed location is converted into human readable format (high level context) from latitude and longitude (low level context), further clicking on the map address pin points the location of a user. In the later case, independent sensors such as medical sensors for collecting a patient’s physiological health information are simulated using a Android device, which can send for example a high blood pressure message to another agent representing a patient.


Insert JSON file

File name Patient.txt

LOAD RULES

The following external context(s) will be added to the working memory.

Tell(1,2,hasBloodPressure(Mary,Normal))
hasGPSLocation(Mary,UNMC)

View device on map

Fig. 7: Reverse geo-coded address

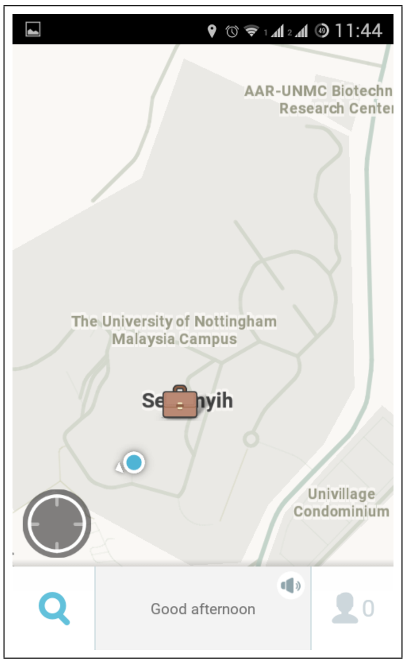

11:44
AAR-UNMC Biotechn Research Center
The University of Nottingham Malaysia Campus
Semenyih
Univillage Condominium
Good afternoon
0

Fig. 8: Address tracked on map

### 4.3 System Startup and Initialization

If we use the desktop interface to validate a set of rules and facts (or initial contexts) while implementing an agent, the resulting JSON file could be used in the Android device for further processing. The JSON files can be copied to Android device via a sync cable, as an email, via bluetooth or as an web service to fetch the files. Rules are stored in a rule-base and facts are stored in the working memory. Since in our system a context-aware rule-based agent uses forward chaining algorithm, it starts with the initial facts stored in the agent’s working memory. Although a set of initial facts is provided by the programmer while creating an agent, it is not always necessary because an agent may use messages received from other agents or sensed data from the environment as initial facts. In our system, an agent’s working memory contains a set of current contexts (facts) and it changes over time by executing matching rule instances. If any update is discovered in the agent’s working memory, the *match-select-act* cycle is invoked to infer new high-level context(s) from the available contexts, and context-aware services are then provided based on the updated contexts. In the following, we discuss *match-select-act* cycle.

## 4.4 Match: Conflict set generation

The rule matching or conflict set generation algorithm generates a set of applicable rule instances according to current contexts or working memory facts. A forward chaining algorithm unifies all antecedents of all rules with a subset of relevant working memory facts. That is, for each rule, the algorithm matches all its antecedents to the facts from the working memory, if all antecedents of a rule are matched then it will check if the consequent is already in the working memory or not, if not then the corresponding rule instance will be added to the conflict set. This will be repeated until no more rule matches. The conflict set may contain more than one rule instance with different priorities. The Algorithm 1 describes the steps involved in conflict set generation.

## 4.5 Select: Conflict resolution

In this phase the conflict between rule instances residing in the conflict set is resolved. Conflict resolution is the order that a rule instance is removed from the agenda or conflict set and its actions executed. In this implementation, we only use *rule ordering strategy* using the rule priority, which is an integer, determines which rule should be executed before the others. The Algorithm 2 describes the steps involved in conflict resolution, which is selecting one rule instance from the conflict set that has the highest priority. If there are multiple rule instances with the same priority exist, the rule instance to be executed is selected randomly.

## 4.6 Act: Execution of the selected rule instance

Execution of a rule instance is straight forward. When the rule instance selected from the conflict set is forwarded for execution, its consequent is added to the working memory as well as processed for further actions depending on the nature of the rule. Consequent of the rule instance can be in a form of communication directive, a fact as a newly derived context simply to be added to the working memory or taking any other action. In order to achieve this, as we have already mentioned, the flag plays an indicator. If the flag is ‘G’, then a goal has been achieved, consequent will be added to the working memory and the system needs to be halt. Similarly the flag ‘C’ indicates that the consequent will be added to the working memory and at the same time the communication part needs to be invoked for this specific execution of rule instance. On the other hand, the flag ‘D’ indicates that the consequent will only be added to the working memory. The communication between the agents are achieved using communication rules. If a rule has either an $Ask$ or a $Tell$ as

**Input: R:** Rule-Base,**WM**: Working Memory
[$\mathbf{R}_s$: A single rule, $\mathbf{R}_i$:A rule instance, $\mathbf{R}_b$: Rule body, $\mathbf{R}_{ib}$: Rule instance body, $\mathbf{R}_c$: Rule consequent, $\mathbf{R}_a$: Rule atoms in the body, $\mathbf{R}_{ap}$: Rule atom predicate, $\mathbf{R}_{at}$: Rule atom terms, $\mathbf{F}_c$: Current fact, $\mathbf{F}_{cp}$: Current fact predicate, $\mathbf{F}_{ct}$: Current fact terms, **PM**: Pattern matching, $\mathbf{P}_{ra}$: Patterns in rule body, **VAR**: Arraylist to hold KEY and VALUE]
**Result:** CS: Conflict set

```
START
for r = 0 to size of R do
    do
        Clear VAR
        $\mathbf{R}_s$ =R[r]
        Find patterns in $\mathbf{R}_b$ of $\mathbf{R}_s$
        Add to Array $\mathbf{P}_{ra}$
        Flag:Array of size equal to $|\mathbf{P}_{ra}|$
        for ra = 0 to size of $\mathbf{R}_a$ do
            Select $\mathbf{R}_a$[ra]
            Seperate $\mathbf{R}_{ap}$ from $\mathbf{R}_{at}$
            for f=0 to size of WM do
                $\mathbf{F}_c$= WM[f]
                Seperate $\mathbf{F}_{cp}$ from $\mathbf{F}_{ct}$
                if $\mathbf{R}_{ap}$==$\mathbf{F}_{cp}$ then
                    if $\mathbf{R}_{at}$==$\mathbf{F}_{ct}$ || pattern($\mathbf{R}_{at}$==$\mathbf{F}_{ct}$) then
                        Add 1 to Flag
                        KEY=$\mathbf{R}_{at}$
                        VALUE=$\mathbf{F}_{ct}$
                        if (VAR does not contian KEY) then
                            Add KEY to VAR
                            Add VALUE to VAR
                        end
                    end
                    else
                        Add 0 to Flag
                    end
                end
            end
        end
        if Flag does not contain 0 then
            for var= 0 to size of VAR do
                Key=VAR[var]
                Value=VAR[var+1]
                $\mathbf{R}_i$= Replace Key with Value in $\mathbf{R}_s$
                $\mathbf{R}_c$=consequent($\mathbf{R}_i$)
                var $\leftarrow$ var+2
            end
            if WM does not contain $\mathbf{R}_c$ && CS does not contain $\mathbf{R}_i$ then
                Add $R_i$ to CS
            end
        end
    while (new matching rule instance found);
end
END
```

**Algorithm 1:** Conflict set generation

```
Input: CS:Conflict set [$P_o$: Priority Operator, SPR: Same priority rules, $C_{ics}$: An element of CS, $R_{ip}$: Rule instance priority ]
Result: to_fire= A selected rule instance to be fired
START
$P_o$=0
for cs = 0 to size of CS do
    $C_{ics}$ = CS[cs]
    get $R_{ip}$ for $C_{ics}$
    if $R_{ip} > P_o$ then
        $P_o$=$R_{ip}$
        $to\_fire$ = $C_{ics}$
    end
    else if $P_o$ == $R_{ip}$ then
        Add $C_{ics}$ to SPR
    end
end
if | SPR | >0 then
    Add to_fire to SPR
    to_fire = select a random instance from SPR
end
END
```

**Algorithm 2:** Conflict resolution

```
Input: to_ fire: A selected rule instance to be fired [$R_c$: A communication rule instance, $R_g$: A rule instance contains a goal context, $R_d$: A deduction rule instance, $R_f$: Rule Flag, $R_{cons}$: Consequent, MAX_SIZE: memory size]
Output: Rule instance executed, consequent added to WM and corresponding action performed.
START
to_ fire from conflict resolution and $R_{cons}$ is the consequent
if $R_g$ then
    if $R_{cons}$ is a conflicting context then
        Overwrite the contradictory context with $R_{cons}$
    end
    else if |WM| < MAX_SIZE then
        Add $R_{cons}$ to WM
    end
    else
        Overwrire an existing context with $R_{cons}$
    end
    Goal Reached
    Execution Halts
end
else
    if $R_{cons}$ is a conflicting context then
        Overwrite the contradictory context with $R_{cons}$
        if $R_c$ then
            initiate communication module
        end
    end
    else if |WM| < MAX_SIZE then
        Add $R_{cons}$ to WM
        if $R_c$ then
            initiate communication module
        end
    end
    else
        Overwrire an existing context with $R_{cons}$
        if $R_c$ then
            initiate communication module
        end
    end
end
END
```

**Algorithm 3:** Execution of a rule

its consequent, we call it a communication rule. Communication rules are handled differently than deduction rules. We discuss agent communication in more detail in Section 4.8. When a rule instance is fired how its consequent is added to the working memory as a newly derived context is discussed in the following section.

## 4.7 Working Memory Updating

The working memory of an agent carries facts which can be initial facts, the newly inferred facts as a result of execution of any rule, or the communicated facts received as messages from other agents. In any case it provides a holder for the available current contexts and to perform context reasoning. In the whole system design and implementation processes where the emphasis is given on the resource constrains, memory is one of the key resources we aim to save. The limit on the size of the working memory is to ensure it does not exceed the maximum number of contexts it can store at any given time, but the facts are generated at almost every iteration and keeping the facts that are more vital to the execution is a crucial task. In our implementation, the working memory is basically a fixed size array. Initially it is empty and once the initial facts are provided, an agent starts context-aware reasoning. The working memory of an agent is divided into static memory and dynamic memory. The dynamic memory is bounded in size, where one unit of memory corresponds to the ability to store an arbitrary context. The static part contains initial facts to start up the system, thus it's size is determined by the number of initial facts. The dynamic part contains newly derived facts as the agent performs context-aware reasoning. Only facts stored in the dynamic memory may get overwritten, and this happens if an agent's memory is full or a contradictory context arrives in the working memory (even if the memory is not full). 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 dynamic memory is full. Because of the bounded dynamic memory, there might be the case when the system can go into an infinite execution if there is no forceful stop, and the goal is not achievable. To overcome this issue we set the number of iteration equal to the number of rules we have to ensure that every rule is checked and in case of no matches are found, instead of abrupt behaviour it will halt itself, saving resources of the host system. The Algorithm 3 describes the steps involved in execution of the selected rule instance and the updating of the working memory.

## 4.8 Communication and Subroutine Handling

Besides the conventional rule firing and updating the working memory facts, the application is also capable of handling different behaviours which are the outcomes of the consequent of a rule instance. For instance, agents can exchange messages regarding their current contexts. In order to achieve this, an agent has to invoke the communication subroutine. Where the communication subroutine is responsible for exchanging the information from one device to another. In [12], $Ask$ and $Tell$ primitives have been defined to achieve communication between agents (e.g., two smart devices), $Ask$ is used when one device asks for some contextual information, similarly $Tell$ is used to answer the ask or simply conveying some contextual information without being asked. However, in practice how the contextual information is sent or received is a matter of question. In our implementation, the devices can communicate in a variety of ways, including Bluetooth, Infrared, Wireless, and SMS. These different modes give these devices diversity in communication. We proposed that in order to achieve an efficient communication, a table has to be maintained and distributed among all the connected devices. This table contains a list of available communication modes supported in the domain. Each device is assigned with a numeric ID, and this ID can be used in the $Ask(i, j, p(t_1, t_2))$ and $Tell(i, j, p(t_1, t_2))$, which will also keep the logical structure of the rule intact. The $i$ and $j$ specify the $FROM$ and $TO$ respectively. If we assign them numbers from the table, it can specify which devices are communicating with each other. For example,

| Agent ID | Bluetooth | IP address | ICCID (Cell number) |
|---|---|---|---|
| 1 | BP monitor | x.y.z.a | 111222333 |
| 2 | Patient care | x.y.z.b | 111222444 |
| 3 | Caregiver | x.y.z.c | 111222555 |

Table 2: Agent ID table

when $i = 2$ and $j = 3$, the $Ask$ primitive becomes $Ask(2, 3, p(t_1, t_2))$, where $p(t_1, t_2)$ is an atomic context which neither contains an $Ask$ nor a $Tell$, and according to the Table 2 where the ID 3 is associated with a caregiver device (as agent 3) and 2 is associated with a patient care device (as agent 2). In case if the patient care agent wants to communicate with the blood pressure monitor agent, it can use the same format by specifying the ID of the blood pressure monitor device. The rest of the columns specify the different available modes of communication and their respective addresses. In case of Bluetooth communication, these devices have to be paired with each other. Once paired names are added to a pair list, they can be specified in the table in order to initiate communication. Once the agents' IDs are specified, the communication mode can be specified explicitly by adding the communication mode at the beginning of the $Ask$ and $Tell$ rule, e.g., $Bluetooth(Ask(i, j, p(t_1, t_2)))$, which will be taken as agent $i$ wants to communicate with agent $j$ using Bluetooth only. In case if no pre-rule communication method is defined then any of the available communication modes can be used. While this is so far handling communication using the rules, but in order to make it work every communication method has to be attached with its respective handler and a method has to be specified which can understand the rule and interpret it into Android specific format. These rules before triggering will be checked with the $Ask$ and $Tell$ rules. If any of them is found, a subroutine will be called to handle the rule, which will extract its FROM and TO from the agent ID table along with the communication addresses and perform the action. The communicated contexts when received by a receiver agent are stored in a buffer before putting them into it's working memory. If the receiving agent is in the middle of the execution it will first complete its current execution and in the next iteration it will add the received contexts from the buffer to it's working memory and will continue further processing.

## 5 A Smart Home Environment Example System

In [27], we have shown how we develop a multi-agent non-monotonic context-aware system whose rules are derived from a smart environment domain ontology. The application is intended to provide care to the patients in smart home environment. Its main design goal is to gather raw data or low

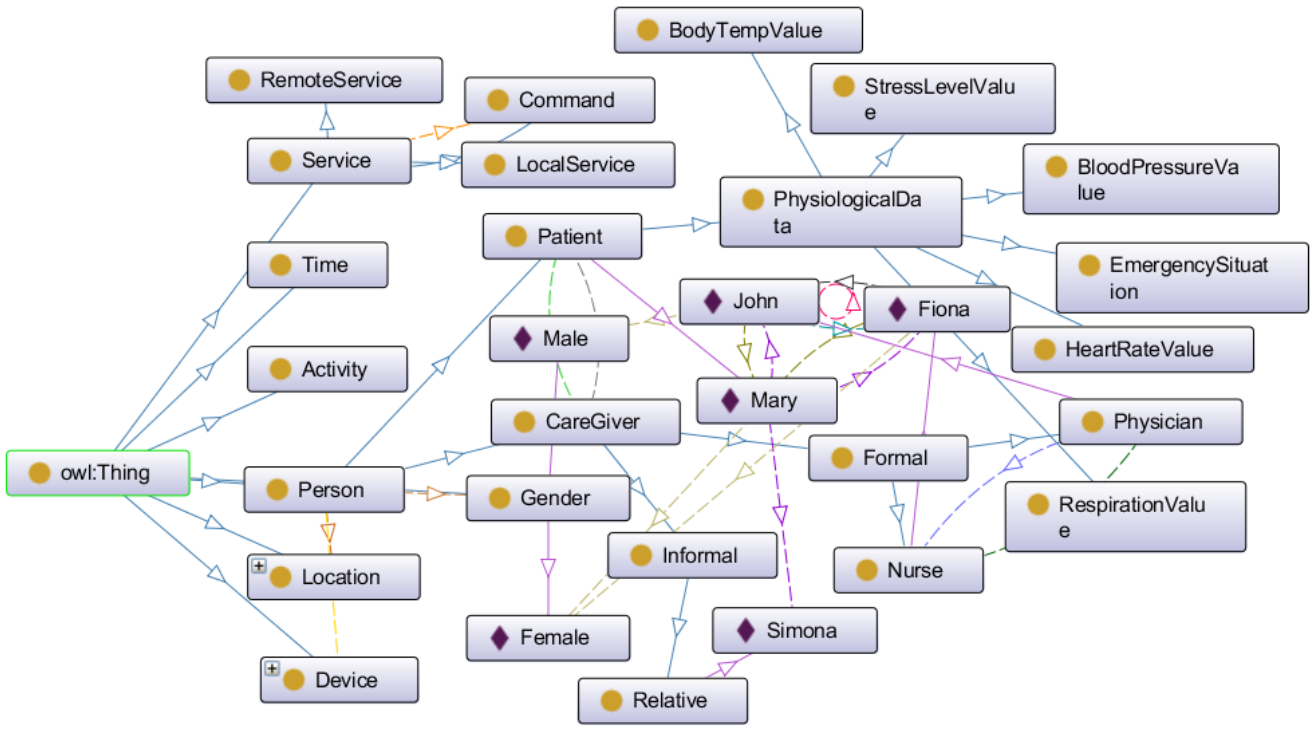


Fig. 9: A fragment of the smart space ontology

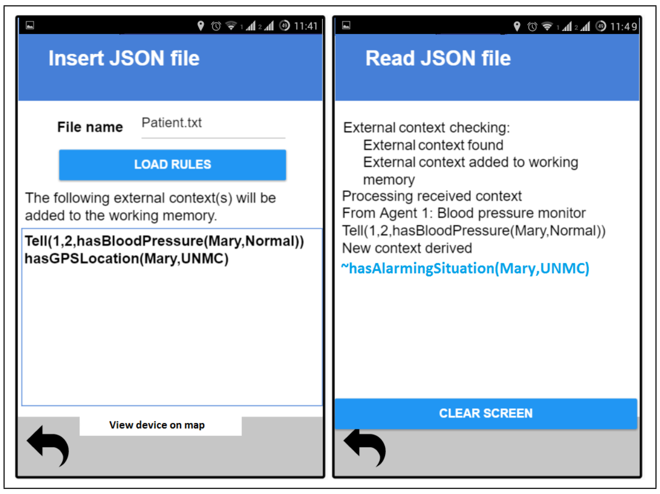
| |
|---|
| **Agent 1: Blood Pressure Agent** |
| **Initial facts:** <br> **Case High:** Person(Mary), SystolicBP (140), DiastolicBP (95), hasSystolicBP (Mary, 140), hasDiastolicBP (Mary, 95), greaterThan(140, 120), greaterThan(95, 80) <br> **Case Normal:** Person(Mary), SystolicBP (118), DiastolicBP (78), hasSystolicBP (Mary, 118), hasDiastolicBP (Mary, 78), greaterThan(118, 90), lessThan(118, 120), greaterThan(78, 60), lessThan(78, 80) |
| 1 : Person(?p), SystolicBP (?sbp), DiastolicBP (?dbp), hasSystolicBP (?p, ?sbp), hasDiastolicBP (?p, ?dbp), greaterThan(?sbp, 120), greaterThan(?dbp, 80) → hasBloodPressure(?p, High) <br> 1 : Person(?p), SystolicBP (?sbp), DiastolicBP (?dbp), hasSystolicBP (?p, ?sbp), hasDiastolicBP (?p, ?dbp), greaterThan(?sbp, 90), lessThan(?sbp, 120), greaterThan(?dbp, 60), lessThan(?dbp, 80) → hasBloodPressure(?p, Normal) <br> 1 : Person(?p), SystolicBP (?sbp), DiastolicBP (?dbp), hasSystolicBP (?p, ?sbp), hasDiastolicBP (?p, ?dbp), lessThan(?sbp, 90), lessThan(?dbp, 60) → hasBloodPressure(?p, Low) <br> 2 : hasBloodPressure(?p, Normal) → Tell(1,2, hasBloodPressure(?p, Normal)) <br> 2 : hasBloodPressure(?p, Low) → Tell(1,2, hasBloodPressure(?p, Low)) <br> 2 : hasBloodPressure(?p, High) → Tell(1,2, hasBloodPressure(?p, High)) |
| **Agent 2: Patient Care Agent** |
| **Initial facts:** Person(Mary), hasPatientID(Mary, P01), PatientID(P01) |
| 1 : Person(?p), hasPatientID(?p, ?pid), PatientID(?pid) → Patient(?p) <br> 2 : Patient(?p), hasBloodPressure(?p, Normal), hasGPSLocation(?p, ?loc) → ∼hasAlarmingSituation(?p, ?loc) <br> 3 : Patient(?p), hasBloodPressure(?p, High), hasGPSLocation(?p, ?loc) → hasAlarmingSituation(?p, ?loc) <br> 3 : Patient(?p), hasBloodPressure(?p, Low), hasGPSLocation(?p, ?loc) → hasAlarmingSituation(?p, ?loc) <br> 4 : Tell(1,2, hasBloodPressure(?p, Normal)) → hasBloodPressure(?p, Normal) <br> 4 : Tell(1,2, hasBloodPressure(?p, High)) → hasBloodPressure(?p, High) <br> 4 : Tell(1,2, hasBloodPressure(?p, Low)) → hasBloodPressure(?p, Low) <br> 5 : Patient(?p), hasAlarmingSituation(?p, ?loc) → Tell(2,3, hasAlarmingSituation(?p, ?loc)) |
| **Agent 3: Caregiver Agent** |
| **Initial facts:** Caregiver(John) |
| 1 : Tell(2,3, hasAlarmingSituation(?p, ?loc)) → hasAlarmingSituation(?p, ?loc) <br> 2 : CareGiver(?c), hasAlarmingSituation(?p, ?loc) → isCaredBy(?p, ?c) |

Table 3: Some rules extracted from the smart space ontology

level context from various sensors that are installed at patient side and in the environment. When the system has adequate data available it can further monitor the patient's situation and in case of patient's discomfort, it seeks the caregiver's attention. In that paper, we have shown how to model a context-aware system based on the logic developed in [12] and how to analyse and formally verify non-conflicting context information guarantees it provides. In this section, we show how we implement an example scenario using Android powered smartphones. The core purpose of modelling domain in the ontology is to contextualize information in an organized and structured way. The set of rules and the set of initial facts are derived from the ontology to model the system considering three agents, namely a Blood Pressure Simulator agent, a Caregiver agent, and a Patient care agent. We list the set of rules that are distributed to these agents in Table 3, which are derived from an ontology. A fragment of the ontology is depicted in Figure 9.

## 5.1 Experimental Setup

In order to implement the verified agents behaviours, we consider three Android powered smartphones representing three agents. The Blood Pressure Simulator agent, which is agent 1, sends either HIGH blood pressure or NORMAL blood pressure status to the Patient care agent 2. After receiving the blood pressure status the Patient care agent performs context-aware reasoning and derives new contexts. It also uses GPS sensor data to derive and add high-level context as its location to the working memory. Based on the context-aware reasoning result, it interacts with agent 3, which is a Caregiver agent. Every agent as per Table 3 performs certain tasks, and its behaviour depends on the rules in its rule-base, the facts provided to it, interaction with other agents as well as information acquired from the environment. In our setup we used two sensors, one is external to the patient care agent i.e., the blood pressure monitor while the other one is a location sensor embedded into the patient care device. The rules designed to implement the system take into consideration the blood pressure of the patient as one of the main decision making factors. As the rules indicate, if the blood pressure is normal, agent 2 displays a non-alarming message situation to the patient. In case of high blood pressure, it informs about the patient's alarming situation to the Caregiver agent. In addition, it also sends the patient's current location acquired from the GPS sensor to the Caregiver agent. Figure 10 (left) depicts that the Patient care agent received a normal blood pressure status message from agent 1, when the patient clicks on the initialize rules, Patient care agent performs context-aware reasoning and derives new contextual information. As we mentioned before, the reasoning engine runs whenever any context is added to the working memory or any update of the context in the working memory is detected. This could be based on firing agent's own rule, receiving a context from other agent or acquiring information from the external environment. After the reasoning process is completed, the screen displays a message to the patient about his status which is depicted in Figure 10 (right). When agent 2 receives a blood pressure status message as high from agent 1 depicted in Figure 11 (left), agent 2 changes its behaviour according to the current context and interacts with the Caregiver agent to inform about the alarming situation of the patient, which is depicted in Figure 11(right). Figure 12 depicts that the caregiver has received the message sent by agent 2 and acted accordingly. Note that information displayed on the screens are only for experimental purposes, in practice the application provides services based on the available contexts and internal operations are hidden from the users. Although this is a small-scale experiment, the results indicate that our system design and implementation is a good choice for practical applications of context-aware computing and resource-bounded practical reasoning.



File name Patient.txt

The following external context(s) will be added to the working memory.

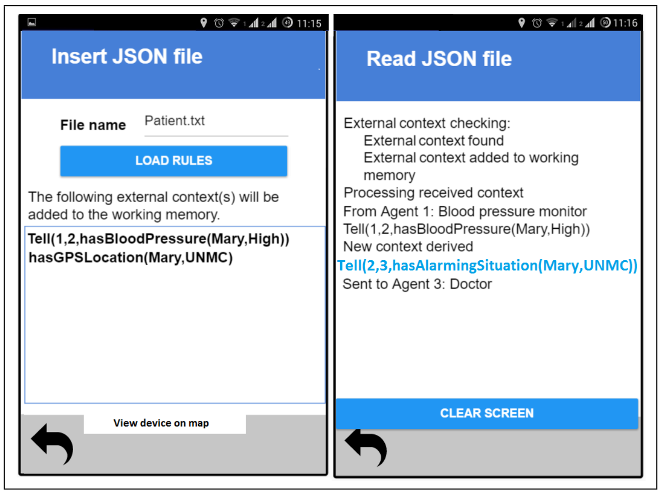
```
Tell(1,2,hasBloodPressure(Mary,Normal))
hasGPSLocation(Mary,UNMC)
```

```
External context checking:
    External context found
    External context added to working
    memory
Processing received context
From Agent 1: Blood pressure monitor
Tell(1,2,hasBloodPressure(Mary,Normal))
New context derived
~hasAlarmingSituation(Mary,UNMC)
```

Fig. 10: Agent 2 displays the normal blood pressure status (The left hand side of the figure shows screen when a message is received from agent 1, and the corresponding context reasoning result shows on the right hand side screen)

## 5.2 Discussion

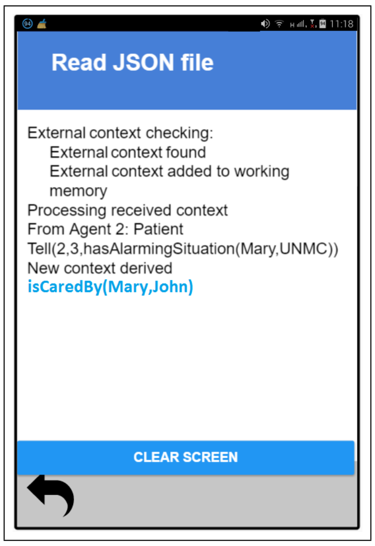
To demonstrate the effectiveness of the formal logical framework presented in [12], we implemented the above mentioned algorithms in Android powered smartphones using the Java language. One of the key features of our approach to rule-based context-aware non-monotonic reasoning is the soundness and completeness of the logical framework compared to many other traditional system design and imple-



The following external context(s) will be added to the working memory.
Tell(1,2,hasBloodPressure(Mary,High))
hasGPSLocation(Mary,UNMC)

External context checking:
External context found
External context added to working memory
Processing received context
From Agent 1: Blood pressure monitor
Tell(1,2,hasBloodPressure(Mary,High))
New context derived
Tell(2,3,hasAlarmingSituation(Mary,UNMC))
Sent to Agent 3: Doctor

Fig. 11: Agent 2 displays the abnormal blood pressure status (The left hand side of the figure shows screen when a message is received from agent 1, and the corresponding context reasoning result shows on the right hand side screen)



External context checking:
External context found
External context added to working memory
Processing received context
From Agent 2: Patient
Tell(2,3,hasAlarmingSituation(Mary,UNMC))
New context derived
isCaredBy(Mary,John)

Fig. 12: Agent 3 displays the fact that caregiver is aware of the emergency situation

mentation approaches. The implemented system guarantees to behave according to its design objective. From the context modelling and contextual reasoning perspectives, a logical language with a clear semantics is used to provide contextual reasoning capabilities of the agents in the system considering knowledge-based reasoning about context in perceptual and sensed data about the real world.

## 6 Conclusions and Future Work

In this paper, we presented and discussed the early development prototype of our resource-bounded context-aware applications based on the logical model developed by [12]. We made a working application of the concept and tested its behaviour on the rules that were used in the verification and the behaviour is found to be the same. The implementation adopted resource friendly mechanism to minimize the use of the memory and processor along with the battery power by restricting the size of the working memories of the agents and their respective message counters. Agents in the application system use rule ordering reasoning strategy while performing contextual non-monotonic reasoning. Here, it is pertinent to mention that the system designer/developer plays a crucial role when designing the rules of the system. The priorities assigned to the rules can make a big difference in the execution of the system. We discussed in detail how communication between the agents is implemented and how the system interprets different kind of rules.

Although it is at a very early stage in the development process, the working prototype showed promising results on a small set of rules. In order to further improve the application, we will implement and study a comprehensive real world context-aware service scenario. The application will be tested against different rule sets with varying sizes and different arrangements to further check its operational behaviour. Furthermore, we would like to enhance the framework considering users' preferences.

**Acknowledgement:** This work is partially supported by the Ministry of Science, Technology and Innovation (MOSTI), Govt. of Malaysia [grant 01-02-12-SF0269].

## References

1. Motorola INC. Motorola demonstrates portable telephone to be availabe for public use by 1976, April 3 1973. Press Release from Motorola Inc.
2. Rafael Ballagas, Jan Borchers, Michael Rohs, and Jennifer G Sheridan. The smart phone: a ubiquitous input device. *Pervasive Computing, IEEE*, 5(1):70–77, 2006.
3. Sebastian Schrittwieser, Peter Frühwirt, Peter Kieseberg, Manuel Leithner, Martin Mulazzani, Markus Huber, and Edgar R Weippl. Guess who's texting you? evaluating the security of smartphone messaging applications. In *19th Annual Network and Distributed System Security Symposium*, 2012.
4. Chunmei Pei, Huiling Guo, Xiuqing Yang, Yangqiu Wang, Xiaojing Zhang, and Hairong Ye. Sensors in smart phone. In *Computer and Computing Technologies in Agriculture IV*, pages 491–495. Springer, 2011.
5. Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen. Contextphone: A prototyping platform for context-aware mobile applications. *Pervasive Computing, IEEE*, 4(2):51–59, 2005.
6. Charith Perera, Arkady B. Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys and Tutorials*, 16(1):414–454, 2014.
7. Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.
8. Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.

9. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 434–441, New York, NY, USA, 1999. ACM.
10. J. E. Bardram, , and N. Nørskov. A context-aware patient safety system for the operating room. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 272–281, 2008.
11. Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.
12. Abdur Rakib and Hafiz Mahfooz Ul Haque. A logic for context-aware non-monotonic reasoning agents. In *Human-Inspired Computing and Its Applications*, pages 453–471. Springer, 2014.
13. Alessandra Esposito, Luciano Tarricone, Marco Zappatore, Luca Catarinucci, Riccardo Colella, and Angelo DiBari. A framework for context-aware home-health monitoring. In *Ubiquitous Intelligence and Computing*, pages 119–130. Springer, 2008.
14. Dejene Ejigu, Marian Scuturici, and Lionel Brunie. An ontology-based approach to context modeling and reasoning in pervasive computing. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on*, pages 14–19. IEEE, 2007.
15. Bin Guo, Daqing Zhang, and Michita Imai. Toward a cooperative programming framework for context-aware applications. *Personal and ubiquitous computing*, 15(3):221–233, 2011.
16. Abdur Rakib and Rokan Uddin Faruqui. A formal approach to modelling and verifying resource-bounded context-aware agents. In *Context-Aware Systems and Applications*, pages 86–96. Springer, 2013.
17. Abdur Rakib, Hafiz Mahfooz Ul Haque, and Rokan Uddin Faruqui. A temporal description logic for resource-bounded rule-based context-aware agents. In *Context-Aware Systems and Applications*, pages 3–14. Springer, 2014.
18. Grzegorz J Nalepa and Szymon Bobek. Rule-based solution for context-aware reasoning on mobile devices. *Computer Science and Information Systems*, 11(1):171–193, 2014.
19. Fabio Sartori, Lorenza Manenti, and Luca Grazioli. A conceptual and computational model for knowledge-based agents in android. *WOA@ AI* IA*, 2013:41–46, 2013.
20. Marco Ughetti, Tiziana Trucco, and Danilo Gotta. Development of agent-based, peer-to-peer mobile applications on android with jade. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM'08. The Second International Conference on*, pages 287–294. IEEE, 2008.
21. Bart van Wissen, Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal. ContextDroid: an expression-based context framework for android. In *Proceedings of the International Workshop on Sensing for App Phones (PhoneSense) 2010*, pages 1–5, 2010.
22. Bc. Ondřej Chrastina. Cross-platform development of smartphone application with the kivy framework. Master's thesis, Masarykova univerzita, Fakulta informatiky, 2015.
23. Simo Hosio, Denzil Ferreira, Jorge Goncalves, Niels van Berkel, Chu Luo, Muzamil Ahmed, Huber Flores, and Vassilis Kostakos. Monetary assessment of battery life on smartphones. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1869–1880. ACM, 2016.
24. H. J. ter Horst. 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.
25. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *WWW2003*, pages 48–57. ACM Press, 2003.
26. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web rule language combining OWL and RuleML. Acknowledged W3C submission, standards proposal research report: Version 0.6, April 2004.
27. Abdur Rakib and Hafiz Mahfooz Ul Haque. Modeling and verifying context-aware non-monotonic reasoning agents. In *Proceedings of the 13th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 453–471. IEEE, 2015.
28. Dana Petcu and Marius Petcu. Distributed jess on a condor pool. In *Proceedings of the 9th WSEAS International Conference on Computers*, pages 1–5, 2005.
29. Wallace Jackson. *Android apps for absolute beginners*. 3rd edition, ISBN13: 978-1-484200-20-9, Apress, Berkeley, California, 2014.
30. Android is the world's largest mobile platform–but it has to overcome these massive hurdles to keep the lead - business insider. `http://www.businessinsider.my/`, Oct 2015.
31. Charles L. Forgy Rete: A Fast Algorithm for the Many Pattern/-Many Object Pattern Match Problem Expert systems, pages 324–341, 1990.